VILNIAUS KOLEGIJA
ELEKTRONIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMINĖS ĮRANGOS KATEDRA

Operacinių sistemų paskaitų konspektas

Parengė: dėst. M. Liogys

Turinys

1. Skaičiavimo sistemos

Skaičiavimo sistema – tai būdų ir priemonių visuma, leidžianti užrašyti, ar kitaip pateikti skaičius skaitmenimis.

Visos skaičių pateikimo sistemos skirstomos į *pozicines* ir *nepozicines.* Nepozicinėse skaičiavimo sistemose simbolio reikšmė nepriklauso nuo jo padėties skaičiuje. Tokių sistemų sudarymo principai nėra sudėtingi. Jų sudarymui naudojamos daugiausiai sudėties ir atimties operacijos.

Pavyzdžiui, sistema su vienu simboliu - pagaliuku buvo naudojama daugelyje tautų. Tam, kad užrašytumėme kažkokį tai skaičių šioje sistemoje, reikėtų užrašyti tam tikrą skaičių pagaliukų, lygų šiam skaičiui. Ši sistema nėra efektyvi, kadangi skaičiaus užrašymas tampa labai ilgas. Kitų nepozicinės sistemos pavyzdžiu galėtų būti romėnų sistema, kurios pagalba skaičiai užrašomi simbolių I, V, X, L, C, D ir kt. dėka. Šioje sistemoje nėra skaitmens nepriklausomybės nuo jo padėties skaičiuje. LX ir XL skaičiuose simbolis X įgauna dvi reikšmes: +10 ir -10.

Pozicinėje skaičiavimo sistemoje skaitmens reikšmė apibrėžiama pagal jo padėtį skaičiuje: vienas ir tas pats ženklas įgauna įvairias reikšmes. Pavyzdžiui, dešimtainiame skaičiuje 222 pirmas skaitmuo iš dešinės reiškia du vienetus, kitas – du dešimtis, o kairysis – du šimtus. Dešimtainė skaičiavimo sistema labiausiai paplitusi skaičiavimo praktikoje.

„Populiarumą“ ji įgijo ne kažkokiomis ypatingomis privilegijomis prieš kitas sistemas, o atsitiktinai – žmogaus rankos turi dešimt pirštu. Kai kuriose tautose pasinaudojama dar ir kojų pirštais, taigi jos naudoja dvidešimtainę skaičiavimo sistemą.

Bet kuri pozicinė skaičiavimo sistema charakterizuojama jos pagrindu. *Pozicinės skaičiavimo sistemos pagrindas (bazė)* – tai ženklų ar simbolių skaičius, naudojamas skaitmenims užrašyti duotoje sistemoje. Todėl gali egzistuoti begalė pozicinių skaičiavimo sistemų, kadangi pagrindu galima imti bet kokį skaičių, taip sudarant naują sistemą.

Pavyzdžiui, šešioliktainė skaičiavimo sistema, kurioje skaičius galima užrašyti simboliais: 0,1, ...,9, A, B, C, D, E, F.

Pozicinėje skaičiavimo sistemoje galioja lygybė

$$A_q=a_{n-1}q^{n-1}+...+a_1q^1+a_0q^0+a_{-1}q^{-1}+...+a_{-m}q^{-m}. \quad (2)$$

Kur A_q – bet koks skaičius, užrašytas skaičiavimo sistemoje, kurios pagrindas q; a_i – skaičiavimo sistemos skaitmuo; n, m – sveikų ir trupmeninių eilių skaičius.

Taigi, skaičių 86,54, remiantis (2) galima užrašyti:

$86{,}54_{10}=8*10^1+6*10^0+5*10^{-1}+4*10^{-2}$. (* žymima sandauga)

Dvejetainėje skaičiavimo sistemoje skaičiams užrašyti naudojami du skaitmenys **0** ir **1.**

Pagal (2) lygybę, dvejetainį skaičių 1001,1101 galime užrašyti taip:

$1001{,}1101_2=1*2^3+0*2^2+0*2^1+1*2^0+1*2^{-1}+1*2^{-2}+0*2^{-3}+1*2^{-4}$.

Jeigu pagal dešimtainės sistemos aritmetikos taisykles atliksime veiksmus su dešinę aukščiau užrašytos lygybės puse, tai gausime dešimtainį duoto dvejetainio skaičiaus $1001{,}1101_2$ ekvivalentą. Jis lygus $9{,}8125_{10}$.

Tam, kad užrašyti vieną ir tą patį skaičių įvairiose skaičiavimo sistemose, reikalingas skirtingas pozicijų arba eilių kiekis. Pavyzdžiui, $96_{10}=140_8=1100000_2$. Kuo mažesnis sistemos pagrindas, tuo didesnis skaičiaus ilgis, t.y. tuo daugiau skilčių.

Jeigu yra duotas skaičiaus ilgis, tuomet ribojama maksimalaus, pagal absoliutinę vertę, skaičiaus reikšmė, kuri gali būti užrašyta. Tegul skaičiaus ilgis lygus bet kuriam teigiamam skaičiui, pavyzdžiui, N. Tada

$max(A_q)=q^{N-1}$. (3)

Pavyzdžiui, skaičiaus ilgis yra 5. Dvejetaine skaičiavimo sistema užrašytas maksimalus skaičius bus $2^4 = 16_{10}$, dešimtaine – $10^4 = 10000$, šešioliktaine – $16^4 = 65536_{10}$.

Jeigu duota maksimali absoliutinė skaičiaus reikšmė, tai skiltinio tinklelio ilgis išreiškiamas:

$N=log_q(max(A_q))+1$. (4)

Pavyzdžiui, maksimali skaičiaus reikšmė yra 4096. Užrašyto dvejetaine skaičiavimo sistema šio skaičiaus ilgis bus $log_2 4096+1 = 13$, šešioliktaine – $\log_{16} 4096+1 = 4$.

Skaičiavimo mašinose apdorojamų skaičių ilgis paprastai apribotas tokiomis reikšmėmis: 1 baitas (8 skiltys), 2 baitai (16 skilčių), 4 baitai (32 skiltys), 8 baitai (64 skiltys). Atitinkamai yra riba ir užrašomiems nurodyto skiltinio tinklelio ilgio sveikiesiems skaičiams.

Pavyzdžiui, maksimalus sveikas teigiamas skaičius, kuris gali buti užrašytas naudojant 16 dvejetainių skilčių, lygus $2^{16}-1=65535$.

Skaičių transformavimas iš vienos skaičiavimo sistemos į kitą. Tarkime turime kažkokį dešimtainį skaičių *A*, norime jį paversti į skaičiavimo sistemą, kurios bazė *q*. Skaičių *A* dalijame iš *q* tol, kol dauginamasis tampa mažesnis už *q*. Gautas liekanas (nuo „paskutinės“ iki „pirmosios“) surašome į vieną eilutę, gautoji eilutė ir bus ieškomas skaičius sistemoje *q*.

1 pavyzdys. Dešimtainį skaičių 2558 perveskime į šešioliktainę skaičiavimo sistemą.

2558 / 16 = 159*16 + 14, (14 = E)

159 / 16 = 9*16 + 15, (15 = F)

9.

Taigi $2558_{10} = 9FE_{16}$.

2 pavyzdys. Dešimtainį skaičių 4796 perveskime į aštuonetainę sistemą.

4796 / 8 = 599*8 + 4,

599 / 8 = 74*8 + 7,

74 / 8 = 9*8 + 2,

9 / 8 = 1*8 +1,

1.

Taigi $4796_{10} = 11274_{8}$.

Tokiu pačiu principu skaičiai iš vienos skaičiavimo sistemos, kurios bazė *q1*, verčiama į skaičiavimo sistemą, kurios bazė *q2*. Reikia atkreipti dėmesį į tai, kad skaičiavimai atliekami pagal *q1* skaičiavimo sistemos taisykles.

Egzistuoja tokios skaičiavimo sistemos, kurių skaičius pervesti į kitos sistemos skaičius paversti labai paprasta (pasinaudojant lentele).

10-ainė	2-ainė	3-ainė	4-ainė	8-ainė	9-ainė	16-ainė
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	10	2	2	2	2	2
3	11	10	3	3	3	3
4	100	11	10	4	4	4
5	101	12	11	5	5	5
6	110	20	12	6	6	6
7	111	21	13	7	7	7
8	1000	22	20	10	8	8
9	1001	30	21	11	10	9
10	1010	31	22	12	11	A
11	1011	32	23	13	12	B
12	1100	40	30	14	13	C
13	1101	41	31	15	14	D
14	1110	42	32	16	15	E
15	1111	50	33	17	16	F

1 lentelė. Skaičiavimo sistemų sąryšis

Tarkime, kad turime skaičių duotojoje sistemoje *q1*, į kitą sistemą *q2* skaičių pervesime pasinaudodami lentele, jei $q1 = q2^n$, arba $q1^n = q2$.

Jei $q1^n = q2$, tai skaičių, duotąjį *q1* sistemoje, susiskirstome į m dalių po n skaitmenų, pradedant nuo dešinės skaičiaus pusės. Jei paskutinėje dalyje n skaitmenų nesusidaro, tai iš

kairės pusės prirašome tiek nulių, kiek jų trūksta iki n. Tuomet kiekvienai daliai užrašome atitinkamą atitikmenį (iš lentelės) *q2* sistemai.

3 pavyzdys. Skaičių 110010110_2 perveskime į šešioliktainę sistemą.

Kadangi $2^4 = 16$, tai duotąjį skaičių suskirstome po keturis skaitmenis (pradedama nuo dešinės).

$110010110_2 = 0001\ 1001\ 0110_2 = 196_{16}$.

4 pavyzdys. Skaičių 112102_3 perveskime į devynetainę sistemą.

Kadangi $3^2 = 9$, tai duotąjį skaičių suskirstome po du skaitmenis.

$112102_3 = 11\ 21\ 02_3 = 472_9$.

Jei $q1 = q2^n$, tai skaičių, duotąjį *q1* sistemoje, susiskirstome į atskirus skaitmenis ir kiekvienam jų užrašome atitikmenis *q2* sistemoje (iš lentelės). Naujojo skaičiaus skaitmenis reikia grupuoti po n (pradedant iš dešinės), jei iki n trūksta iš kairės pusės pridedame nulius.

5 pavyzdys. Skaičių 1708_{16} užrašykime 4- ainėje sistemoje.

Kadangi $16 = 4^2$, tai naujojo skaičiaus skaitmenis grupuosime po 2.

$1708_{16} = 10\ 13\ 00\ 20_4 = 10130020_4$.

6 pavyzdys. Skaičių $166D_{16}$ užrašykime 2- ainėje sistemoje.

Kadangi $16 = 2^4$, tai naujojo skaičiaus skaitmenis grupuosime po keturis.

$166D_{16} = 1\ 0110\ 0110\ 1101_2 = 1011001101101_2$.

Bendros sąvokos

Operacinė sistema (OS) – organizuotas rinkinys programų, skirtų skaičiavimo resursams valdyti, skaičiavimo procesams planuoti ir organizuoti.

Duomenys – bet kokie informaciniai objektai skirti apdoroti ir saugoti ESM. Panašūs duomenys grupuojami (pagal struktūrą, taikymo sritį ar kitus kriterijus) į aibes, vadinamas duomenų rinkiniais. Norėdamas apdoroti duomenis, ESM vartotojas turi pateikti *užduotį* ESM. Užduotį sudaro žingsniai; per vieną žingsnį vykdoma viena vartotojo arba sisteminė apdorojimo programa. Užduoties žingsnio atlikimas – *procesas*, kurį pradeda operacinė sistema. Užduoties apraše nurodomi bendri visos užduoties duomenų rinkiniai ir kiti skaičiavimo resursai; be to, jie detalizuojami kiekvienam užduoties žingsniui. OS valdo būtent užduočių atlikimą.

Skaičiavimo resursai – tai aparatūrinės, programinės bei informacinės priemonės užduočiai atlikti. Mašina iš karto gali spręsti keletą nepriklausomų užduočių, ir visoms joms

reikia skaičiavimo resursų, kurie yra riboti. OS turi valdyti skaičiavimo resursus taip, kad vartotojų užduotys butų atliekamos kuo efektyviau. Atliekamos užduotys nuosekliai pereina parengties, atlikimo ir baigties fazes. Parengties fazėje OS priskiria skaičiavimo resursus užduočiai atlikti. Tai atlieka OS programų valdomi sisteminiai procesai. Parengties fazė baigiasi sudarius proceso, kurį valdys pirmojo užduoties žingsnio programa, aprašą. Atlikimo fazėje sprendžiami užduoties žingsnių uždaviniai – vykdoma pagrindinė užduoties funkcija. Ši fazė baigiasi įvykdžius vartotojo programą. Baigties fazėje resursai grąžinami laisvų resursų fondui, išvedami rezultatai ir kt. Šios fazės, kaip ir parengties, uždavinius atlieka sisteminiai procesai. Paprastai ESM tuo pačiu metu atlieka keletą užduočių, ir įvairios užduotys gali buti įvairiose fazėse, t.y. ESM gali buti sprendžiama kartu keletas sisteminių ir probleminių uždavinių. OS veikia, perjungdama centrinį apdorojimo įtaisą – centrinį procesorių (ar keletą jų) – pasirinktam procesui atlikti. Procesorius perjungiamas, kai pertraukiamas vykstantis procesas.

Registrai – vietinės centrinio procesoriaus atminties įtaisai, kuriose fiksuojama ESM operatyvinio valdymo informacija.

Pertraukimas – tai laikinas vykstančio proceso sustabdymas, norint pereiti prie kito proceso, kurį būtina arba tikslinga atlikti tuo metu.

2. Operacinių sistemų paskirtis ir funkcijos

Pagrindinė operacinės sistemos funkcija – valdyti skaičiavimo resursus. Skaičiavimo resursai – skaičiavimo mašinų mokslo, paprastai vadinamo skaičiavimo technika, objektas. Skaičiavimo resursai – tai visuma techninių, programinių ir informacinių priemonių, organizuojančių ir valdančių skaičiavimo procesą bei atliekančių pačius skaičiavimus. Tai: operaciniai resursai (procesoriai, informacijos mainų kanalai ir kiti panašūs įtaisai);

Techninės priemonės: procesoriai; atminties įtaisai; terminaliniai informacijos įvedimo ir išvedimo įtaisai (spausdinimo įtaisai, skeneriai, displėjai ir pan.); ryšio priemonės (ryšio linijos ir aparatūra informacijai tarp toli esančių įtaisų persiųsti).

Programinės įrangos priemonės (algoritminės kalbos, jų transliatoriai, įvairios kitos sisteminės bei vartotojų programos);

Kalbant apie operacines sistemas, reikia paminėti ir jos vartotojus, naudojančius ESM savo praktinėje veikloje. Tai – programuotojai ir aptarnaujantysis personalas. Programuotojų būrys gana gausus: pirmiausia tai profesionalūs ir neprofesionalūs programuotojai.

Profesionalių programuotojų veiklos sritis – bendrosios programinės įrangos kūrimas, priežiūra ir taikymas konkrečiai ESM. Dabartiniu metu, vis plačiau taikant ESM, daugėja ir neprofesionalių programuotojų. Šie programuotojai savo praktinius uždavinius sprendžia nesikreipdami į profesionalus, ankščiau buvusius tarpininkus tarp vartotojo ir ESM. Šių programuotojų ypač pagausėjo atsiradus asmeninėms ESM. Efektyvi ir „draugiška“ („draugiška“ ta prasme, kad operatyviai padeda vartotojui) programinė ESM įranga, visu pirma „draugiška“ operacinė sistema – sėkmingo šių programuotojų darbo ESM garantas.

Aptarnaujančiam personalui OS sudaro sąlygas lengvai kaupti duomenis apie tai, kaip naudojami skaičiavimo resursai, paprastai keisti ESM konfigūraciją sutrikus aparatūrai ir kitais atvejais, iš anksto paruošti informacijos laikmenų tomus su duomenų rinkiniais ir kt. Palaikydamos ryšį su ESM operatoriumi, OS sudaro sąlygas operatyviai valdyti užduočių atlikimą.

OS gali padidinti skaičiavimo resursų efektyvumą, didindama ESM pralaidumą bei parengties koeficientą ir trumpindama atsako laiką.

ESM *pralaidumas* – atliekamo darbo apimtis per apibrėžtą laiką, pavyzdžiui, atliekamų užduočių skaičius per parą, pamainą ar pan. Jis priklauso tiek nuo aparatūros charakteristikų (atliekamų operacijų skaičiaus per sekundę ir kt.), tiek nuo programinės įrangos kokybės.

Atsako laikas – sistemos (ESM) reakcijos į vartotojo užklausą laikas.

Parengties koeficientas atspindi sistemos pasirengimą atlikti vartotojų užduotis. Šis koeficientas mažas, kai dažnai genda ESM aparatūra ir modifikuojama programine įranga. Jis padidėja prijungus papildomą aparatūrą ir įtraukus į OS klaidų aptikimo procedūras.

Koks svarbus OS vaidmuo didinant skaičiavimo resursų efektyvumą, matyti ir iš to, kaip didėja parengiamųjų darbų laiko ir sprendimo laiko santykis didėjant centrinio procesoriaus našumui.

Parengiamieji darbai – tai operatoriaus atliekami veiksmai pereinant prie naujos užduoties: duomenų rinkinių montavimas įvedimo ir išvedimo ar išorines atminties įtaisuose, įvairūs aparatūros perjungimai ir kt.

Von Neuman kompiuterio architektūra ir veikimo principai.

Dauguma šiandien dar naudojamų ir gaminamų kompiuterių sukurti remiantis pagrindiniais principais, kuriuos prieš daug metų suformulavo von Neuman, Burks ir Goldstine. Nors terminas "von Neumano kompiuteris" seniai žinomas ir įprastas, tikslaus jo apibrėžimo nėra.

Autoriai savo idėją suformulavo taip:

Principinės mašinos komponentės yra:

1) kadangi tai turi būti bendros paskirties skaičiavimo mašina, joje turi būti tokie įtaisai, kaip aritmetinis, atminties, valdymo ir ryšio su žmogumi; mašina bus pilnai automatine;

2) mašina turi saugoti atmintyje tokia pačia forma ne tik skaičiavimams reikalingą skaitmeninę informaciją (duomenis, funkcijų lenteles, tarpinius rezultatus), bet ir komandas, valdančias skaičiavimą; skaičiavimo mašinoje turi būti įtaisas programoms saugoti, o taip pat įtaisas, kuris jas suprastų ir valdytų jų vykdymą;

3) konceptualiai mes apsvarstėme dvi skirtingas atminties formas - skaičių ir komandų atmintį; jei komandos bus koduojamos skaičiais ir jei mašina sugebės atskirti jas nuo skaičių, viskas gali būti saugoma vienoje atmintyje;

4) jei atmintis tik saugo komandas, mašinoje turi būti organas, kuris automatiškai jas vykdo. Šį organą vadinsime valdymu;

5) kadangi tai yra skaičiavimo mašina, joje turi būti aritmetinis įtaisas, kuris vykdytų aritmetines operacijas (sudėtį, atimtį, daugybą, dalybą), o taip pat ir kitas dažnai sutinkamas operacijas;

6) joje taip pat turi būti įvedimo ir išvedimo įtaisas, kuris leistų operatoriui komunikuoti su mašina.

Von Neumano mašinos architektūrą sudaro tokie įtaisai:

1) **CPU**, apjungiantis operacinį (duomenų procesorių) ir valdymo (komandų procesorių) įtaisus; CPU interpretuoja ir vykdo programos komandas (išskyrus įvedimo ir išvedimo komandas);

2) **atmintis**, kurioje saugomi visi duomenys ir programos;

3) **įvedimo ir išvedimo įtaisas**, kuris kartu su periferiniais įtaisais užtikrina kompiuterio ryšį su aplinka;

4) vidiniai duomenų keliai - **magistralės**, kurios užtikrina informacijos mainus tarp visų kompiuterio įtaisų.

Mažiausias identifikuojamas informacijos vienetas kompiuteryje - dvejetainis vektorius. Šis vektorius von Neumano kompiuteryje atitinka tokius informacijos tipus:

- **komandas**,
- **duomenis** (skaičius, dvejetainius vektorius ar simbolius),
- atminties ląstelių arba įvedimo ir išvedimo įtaisų **adresus**.

Šiuolaikiniuose kompiuteriuose (ne von Neumano tipo) galima sutikti ir kitokius informacijos tipus:

- **tegus** (*tags*) - bitų grupes, kurios nurodo palydimos informacijos tipą;
- informacijos vienetų **deskriptorius**;

• informacijos vienetų **identifikatorius** (vardus).

Von Neumano kompiuteryje atminties ląstelės turinys interpretuojamas pagal tokią schemą:

a) pirmiausia kompiuteris pagal savo būseną nustato, kuriame interpretacijos žingsnyje jis yra, t.y., ar šis žodis turi būti interpretuojamas kaip komanda, ar kaip duomenys;

b) duomenų interpretaciją apsprendžia vykdomos komandos tipas (operacijos kodas); šios komandos argumentu ir yra duomenys.

Von Neumano kompiuterio komandos vykdymo procesas susideda iš dviejų fazių: pirmojoje fazėje išrenkama ląstelė, kurios turinys bus interpretuojamas kaip komanda. Antrojoje fazėje ši komanda vykdoma, pagal komandoje esančią informaciją iš atminties išrenkant ląstelę, kurios turinys interpretuojamas kaip apdorojamas operandas.

Detaliau nagrinėjant komandos vykdymo procesą, galima išskirti daugiau smulkesnių žingsnių, pavyzdžiui:

1. Išrinkti iš atminties komandą ir įrašyti ją į komandos registrą.
2. Pakeisti komandos skaitiklio, kuris nurodo vykdomos komandos adresą, turinį.
3. Nustatyti išrinktos komandos tipą.
4. Jei komandoje nurodyti duomenys yra atmintyje, nustatyti jų vietą.
5. Jei reikia, išrinkti iš atminties duomenis ir perduoti juos į CPU registrus.
6. Vykdyti komandoje nurodytą operaciją.
7. Rezultatus įrašyti į nurodytą vietą.
8. Pereiti į 1-ą žingsnį kitai komandai vykdyti.

Operacinių sistemų tobulėjimui didelę įtaką turėjo, tobulėjanti technika.

Dž. Ekertas ir Dž. Močli (Pensilvanijos universitetas) JAV kariškių pavedimu dar karo metais sukūrė pirmąjį elektroninį bendrosios paskirties kompiuterį **ENIAC** (Electronic Numerical Integrator and Calculator). Šis kompiuteris buvo skirtas artilerijos šaudymo lentelėms skaičiuoti. U formos kompiuteris buvo 24 metrų ilgio, 2,5 m aukščio ir svėrė apie 30 tonų. Viso kompiuteryje buvo panaudota 18000 elektroninių lempų. Tai buvo programuojamas kompiuteris. Tiesa, programavimas buvo atliekamas rankiniu būdu, atitinkamai sujungiant kabelius ir nustatant perjungėjus. Duomenys buvo įvedami naudojant perfokortas. Skaičiavimams reikalingų programų "įvedimas" trukdavo net visą dieną.

1944 m. Von Neumanas buvo pakviestas dirbti prie ENIAC projekto. Siekdami supaprastinti programavimą, projekto autoriai parengė atmintyje saugomos programos

panaudojimo ideją. Šis pasiūlymas buvo paskelbtas ir davė pradžią plačiai žinomai sąvokai "Von Neumano kompiuteris".

Pirmuoju kompiuteriu, kuriame programa saugojama jo atmintyje, laikomas 1949 m. M. Uilkso realizuotas **EDSAC** (Electronic Delay Storage Automatic Calculator) projektas Kembridže.

Pirmuoju komerciniu kompiuteriu laikomas **UNIVAC I** (1951 m.). Jis kainavo apie 1 mln. dolerių. Tokių kompiuterių buvo pagaminta 48.

IBM firma, kuri buvo žinoma kaip perfokortų ir įstaigų automatizavimo įrangos gamintoja, 1950 m. įsijungė į kompiuterių gamybą. Jos pirmasis kompiuteris buvo IBM 701, kurių pagaminta 19. Tais laikais vyravo nuomonė, kad kompiuteris - labai specializuotas įtaisas, turintis siaurą rinką. Šią nuomonę sulaužė IBM, investavusi 5 mlrd. dolerių ir 1964 m. paskelbusi apie sistemą **IBM/360**, kurią tuo metu sudarė 6 modeliai, besiskiriantys savo kaina ir našumu iki 25 kartų.

1965 m. DEC pristatė PDP-8 - pirmąjį komercinį **minikompiuterį** - mažą ir palyginus pigią (apie 20 000 dol.) mašiną. Minikompiuteris laikomas šiuolaikinių mikroprocesorių protėviu.

1973 m. buvo pristatytas pirmasis **superkompiuteris** - CDC 6600. Jo autorius S.Krėjus vėliau paliko CDC ir įkūrė savo firmą (Cray Research), kurioje 1976 m. sukūrė Cray-1, kuris tuo metu buvo greičiausias ir brangiausias pasaulyje, tačiau moksliniams uždaviniams turėjo geriausią našumo ir kainos santykį.

Po metų, 1977 m., S.Džobs ir S.Vozniakas davė pradžią **personalinių kompiuterių** pramonei, sukurdami Apple II. Tačiau po 4 metų IBM perėmė personalinių kompiuterių gamybos vairą į savo rankas. Dabar IBM ar IBM tipo personaliniai kompiuteriai tvirtai vyrauja rinkoje.

3. Duomenų vaizdavimas atmintyje.

Kompiuterio atmintis yra sudaryta iš milijonų tranzistorių, naudojamų kaip elektroniniai jungikliai. Konkrečiu laiko momentu kiekvienas jungiklis yra įjungtas arba išjungtas, atitinkamai įtampa yra arba nėra. Rinkiniai "įtampa yra/nėra" naudojami fiziniam duomenų saugojimui kompiuterio atmintyje. (Magnetinėse informacijos talpyklose vietoje įtampos naudojamas skirtingas įmagnetinimas.) Kiekvienas toks jungiklis atitinka konceptualiai mažiausią saugomos informacijos vienetą, kuris vadinamas **bitu**. Dvi jo būsenos gali būti

interpretuojamos kaip dvejetainiai skaičiai (skaitmenys) 1 ir 0 atitinkamai. Pats žodis "bitas" yra angliškų žodžių "dvejetainis skaitmuo" santrumpa (*angl. bit - binary digit*).

Aštuonių bitų grupė vadinama **baitu** (*angl. byte*).

1 baitas = 8 bitai

Naudojami ir didesni matavimo vienetai - **K baitų**, **M baitų**, **G baitų**, **T baitų** (žymima KB, MB, GB ir TB atitinkamai):

1 KB = 1024 baitai
1 MB = 1024 KB = 1 048 576 baitai
1 GB = 1024 MB = 1 048 576 KB = 1 073 741 824 B
1 TB = 1024 GB = 1 048 576 MB = 1 073 741 824 KB = 1 099 511 627 776 B

Dviejų ar daugiau baitų grupė dažnai vadinama **žodžiu** (*angl. word*). Žodžio dydis priklauso nuo naudojamo kompiuterio. Paprastumo dėlei žodį laikysime dviem baitais.

Visi duomenys kompiuterio atmintyje vaizduojami bitų rinkiniais. Pirmiausia panagrinėsime skaičių vaizdavimą. Egzistuoja keletas skaičių vaizdavimo formatų. Formatas, kuriame dešimtainio (dvejetainio) taško pozicija yra fiksuota, vadinamas fiksuoto taško formatu. Sveikiems skaičiams galima įsivaizduoti, kad dešimtainis (dvejetainis) taškas yra po paskutinio skaitmens, todėl dažniausiai jie vaizduojami šiuo formatu.

Kiek baitų skiriama sveikam skaičiui, priklauso nuo duomenų tipo (sveikas skaičius, ilgas sveikas skaičius ar trumpas sveikas skaičius) ir nuo naudojamo kompiuterio.

Norint sveiką neneigiamą skaičių pavaizduoti kompiuterio atmintyje dviejuose baituose, reikia jį užrašyti dvejetainėje skaičiavimo sistemoje ($5 = 101_2$) ir papildyti nuliais iš kairės, kad būtų 16 dvejetainių skaitmenų:

5 00000000 00000101

Sveiki neneigiami skaičiai dar vadinami sveikais skaičiais be ženklo. Akivaizdu, kad mažiausias sveikas skaičius be ženklo yra 0:

0 00000000 00000000

O didžiausias sveikas skaičius be ženklo, kurį galima pavaizduoti 2 baituose, yra 65 535 = $2^{16} - 1$:

65 535 11111111 11111111

Analogiškai viename baite gali būti pavaizduoti sveiki skaičiai be ženklo nuo 0 iki 255, o keturiuose baituose - nuo 0 iki 4 294 967 295 ($2^{32} - 1$).

Vaizduojant sveikus skaičius su ženklu, galima būtų kairiausią bitą skirti ženklo vaizdavimui: 0 reikštų pliusą, o 1 - minusą. Tada dviejuose baituose galima būtų pavaizduoti skaičius nuo +0 iki +32 767 ir nuo -0 iki - 32 767, tačiau tokiu atveju 0 būtų vaizduojamas

dviem skirtingais būdais ir reikėtų specialios aritmetikos, dirbančios su dviem nuliais. Todėl neneigiamų skaičių vaizdavimui naudojamas tiesioginis kodas (vaizduojamas pats skaičius x, t.y. vaizduojamas taip pat kaip ir skaičius be ženklo, tik vienas bitas rezervuojamas ženklui), o neigiamų skaičių vaizdavimui naudojamas **papildomas kodas** (vietoje skaičiaus -x vaizduojamas skaičius 2^N - x, kur N - skaičius bitų, skiriamų skaičiui pavaizduoti). Papildomas kodas gali būti gautas tokiu būdu:

1. *Pavaizduojamas atitinkamas teigiamas skaičius.*
2. *Visi bitai invertuojami, t.y. 0 keičiamas 1 ir atvirkščiai.*
3. *Pridedamas 1.*

Panagrinėkime, pavyzdžiui, kaip bus pavaizduotas skaičius -5:

1. 5 00000000 00000101
2. Bitai invertuojami 11111111 11111010
3. Pridedamas 1 11111111 11111011

Naudojant tokį vaizdavimo formatą, visų neneigiamų skaičių kairiausias bitas yra 0, o neigiamų - 1. Dviejuose baituose galima pavaizduoti sveikus skaičius su ženklu nuo -32 768 iki 32 767:

0	00000000 00000000
1	00000000 00000001
…	…
32 766	01111111 11111110
32 767	01111111 11111111
-32 768	10000000 00000000
-32 767	10000000 00000001
…	…
-2	11111111 11111110
-1	11111111 11111111

Analogiškai viename baite gali būti pavaizduoti sveiki skaičiai su ženklu nuo -128 iki 127, o keturiuose baituose - nuo -2 147 483 648 iki 2 147 483 647.

Sveiki skaičiai kompiuterio atmintyje vaizduojami tiksliai, bet jų kiekis nėra didelis. Be to, dirbant su sveikais skaičiais, pačiam programuotojui reikia rūpintis, kad operacijos rezultatas patektų į sveikų skaičių apibrėžimo sritį. Priešingu atveju galima gauti iš pirmo žvilgsnio "keistus" rezultatus. Pavyzdžiui, matematikoje dviejų teigiamų skaičių suma visada yra teigiamas skaičius, tačiau kompiuteriui

32 767 + 1 = -32 768

Tai galima suprasti tik žinant, kaip sveiki skaičiai vaizduojami kompiuterio atmintyje:

+	32 767	01111111 11111111
	1	00000000 00000001
	-32 768	10000000 00000000

Pastaba. Fizinis duomenų vaizdavimas kompiuterio atmintyje gali skirtis nuo aptarto dėl skirtingo baitų išdėstymo. Personaliniuose kompiuteriuose jaunesnieji baitai talpinami atmintyje mažesniais adresais, todėl fiziškai atmintyje sveikas skaičius 5 dviejuose baituose bus pavaizduotas ne taip:

5 00000000 00000101

o šitaip:

5 00000101 00000000

Dabar kompiuteriai daug dirba ne tik su skaičiais, bet ir su tekstais (simboliais). Norint simbolius vaizduoti kompiuterio atmintyje, reikia sudaryti simbolių kodų lentelę, kuri vienareikšmiškai susietų visus reikalingus simbolius su bitų rinkiniais. Kyla klausimas, kiek bitų reikia simbolių vaizdavimui. Anglų kalboje yra 26 didžiosios ir 26 mažosios raidės, 10 skaitmenų ir apie 35 specialius simbolius, naudojamus skyrybai ir pan. - iš viso apie 100 simbolių. Be to, dar reikia specialių "valdančių simbolių", kurie turi specialią prasmę tekstų redaktoriams (pavyzdžiui, nutrinti simbolį) arba kompiuterio įrenginiams (pavyzdžiui, cyptelti). Šešių bitų nepakanka ($2^6 = 64$), taigi minimaliam kodui reikia 7 bitų ($2^7 = 128$). Dauguma kodų naudoja 8 bitus, įtraukdami daugiau specialių, grafinių simbolių arba kontrolei, todėl tradiciškai kompiuteriai skiria 8 bitus, t.y. vieną baitą, simbolių vaizdavimui.

Dauguma šiuolaikinių kompiuterių sistemų naudoja tarptautinės standartų organizacijos ISO simbolių kodų lenteles. Amerikietiškas variantas (ANSI X3.41977) vadinamas ASCII (*American Standard Code for Information Interchange*). ASCII yra 7 bitų kodas, apibrėžiantis pirmąją simbolių lentelės pusę - 128 simbolius su kodais nuo 0 iki 127. Kitos kalbos turi raidžių, kurių nėra anglų kalboje, bet siekiant, kad pirma simbolių lentelės pusė būtų vienoda, jos talpinamos antroje lentelės pusėje (su kodais nuo 128 iki 255).

Apžvelkime ASCII kodų lentelę. Trijų pagrindinių grupių - skaitmenys, didžiosios raidės ir mažosios raidės - simboliai grupėse eina iš eilės be praleidimų:

0 - 48, 1 - 49, … 8 -56, 9 - 57;

A - 65, B - 66, … Y - 89, Z - 90;

a - 98, b - 99, … y - 121, z - 122.

Pirmieji 32 simboliai (kodai 0 - 31) yra specialūs "valdantys simboliai". Pavyzdžiui, simbolis su kodu 13 reiškia karietėlės grąžinimą (*angl. Carriage Return*), simbolis su kodu 10 - perėjimą į naują eilutę (*angl. Line Feed*). Kad suvoktumėme šių simbolių prasmę, reikia prisiminti, kaip elgiamasi su mechanine spausdinimo mašinėle baigus eilutę: karietėlė atstumiama į eilutės pradžią ir pasukama (paspaudžiama) rankenėlė, kad pereitumėme į naują eilutę. Tradiciškai daugeliui tekstų redaktorių, dirbančių operacinės sistemos MS-DOS

aplinkoje, tokia simbolių pora (su kodais 13 ir 10) reiškia eilutės pabaigą. UNIX operacinėse sistemose, eilutės pabaiga žymima vienu simboliu su kodu 10.

Kiti specialūs simboliai - skirtukai, operacijų ženklai, skliaustai ir pan. - išbarstyti likusiose vietose. Pavyzdžiui, simbolio " " (tarpas) kodas 32, "[" (atidarantis laužtinis skliaustelis) - 91, "{" (atidarantis riestinis skliaustelis) - 123.

Kyla klausimas, kokius dar duomenis galima pavaizduoti bitų rinkiniais. Jei kiekvienam 1 suteikti prasmę "yra", o 0 - nėra, tai 8 bitų rinkinys galėtų būtų suprastas kaip pica su tokiais ingredientais, išvardintais eilės tvarka: dešra, sūris, kumpis, alyvuogės, grybai, ananasai, saldieji pipirai, svogūnai.

Tada 01100010 išreikštų picą su sūriu, kumpiu ir saldžiaisiais pipirais, tačiau be dešros, alyvuogių, grybų, ananasų ir svogūnų.

Iš tikrųjų bitų rinkiniai gali vaizduoti bet kokius duomenis ir kompiuteris nieko nežino apie jų žmogišką interpretaciją, jis tiesiog saugo juos ir atlieka nurodytas instrukcijas. Tas pats bitų rinkinys gali išreikšti visiškai skirtingus duomenis. Pavyzdžiui, 01100010 gali išreikšti:

1. - picą su sūriu, kumpiu ir saldžiaisiais pipirais,
2. - raidę "a",
3. - sveiką skaičių 98.

Kiekvieno tipo duomenims vaizduoti skiriamas baigtinis bitų skaičius, o kiekvienas bitas gali įgyti tik vieną iš dviejų reikšmių, todėl galima pavaizduoti tik baigtinę reikšmių aibę, t.y. gali būti vaizduojami tik diskretiniai duomenys. Tai reiškia, kad kompiuteryje negalima pavaizduoti, pavyzdžiui, visų atkarpos taškų.

Dėl šių priežasčių kyla problemos, vaizduojant kompiuterio atmintyje realius skaičius. Tarp bet kurių dviejų realių skaičių yra be galo daug realių skaičių, tačiau jie visi negali būti pavaizduoti kompiuterio atmintyje. Tačiau reikia turėti galimybę atlikti skaičiavimus, kuriuose operuojama su plataus spektro realiais skaičiais: tiek su labai mažais, pavyzdžiui, elektrono masė; tiek su labai dideliais, pavyzdžiui, atstumas iki žvaigždės. Šių problemų sprendimui pravers mokslinis formatas (*angl. scientific notation*): kiekvieną skaičių galima užrašyti tokia forma

$$x = \langle\text{ženklas}\rangle\ \langle\text{mantisė}\rangle * P^{\langle\text{eilė}\rangle}$$

kur ženklas - "+" arba "-", mantisė yra neneigiama, o P yra skaičiavimo sistemos pagrindas. Pastebėkime, kad toks užrašas nėra vienareikšmis - keičiant taško poziciją mantisėje ir tuo pačiu metu eilės reikšmę, pats skaičius nesikeičia, pavyzdžiui:

$-1.75 * 10^{0} = -17.5 * 10^{-1} = -0.175 * 10^{1}$

Dėl vienareikšmiško apibrėžtumo dažniausiai pridedami papildomi reikalavimai mantisei, kad ji būtų normalizuota. Normalizuotos mantisės sąvoka priklauso nuo kompiuterio architektūros. Kai kuriuose kompiuteriuose nenulinio skaičiaus mantisė yra mažesnis už 1 skaičius, kurio pirmas skaitmuo yra nenulinis, t.y.

$$x = \langle ženklas \rangle\ 0.\langle mantisė \rangle * P^{\langle eilė \rangle}$$

Tokiame užraše eilė gali būti ir teigiama, ir nulinė, ir neigiama. Kompiuterio atmintyje dažniausiai vaizduojama ne pati eilė, bet charakteristika - neneigiamas skaičius, gaunamas prie eilės pridėjus konstantą. Ši konstanta ir eilės skaičiavimo sistemos pagrindas priklauso nuo kompiuterio architektūros.

Kadangi dešimtainio (dvejetainio) taško pozicija nėra fiksuota, todėl šis vaizdavimo formatas vadinamas slankaus kablelio (*angl. floating-point*) skaičių formatu.

Toliau nagrinėsime realių skaičių vaizdavimą konkrečiai personaliniuose kompiuteriuose.

Personaliniuose kompiuteriuose eilė yra dvejetainė (P=2) ir pirmasis mantisės skaitmuo visada turi būti 1, todėl naudojama kiek kitokia skaičiaus forma

$$x = \langle ženklas \rangle\ 1.\langle mantisė \rangle * 2^{\langle charakteristika \rangle}$$

kur charakteristika yra sveikas neneigiamas skaičius, gaunamas prie eilės pridėjus konstantą. Šios konstantos dydis priklauso nuo dydžio atminties, skiriamo skaičiui (tame tarpe ir jo charakteristikai) pavaizduoti, pavyzdžiui, kai realiam skaičiui skiriami 4 baitai, charakteristika yra eilė + 127.

Panagrinėkime pavyzdį. Tarkime, kad personalinio kompiuterio atmintyje norime pavaizduoti skaičių -1.75. Pirmiausia jį reikia užrašyti dvejetainėje skaičiavimo sistemoje

$-1.75 = -(1 + 1/2 + 1/4) = -1.11_2$

Tada užrašome dvejetainiu moksliniu formatu taip, kad sveikoji dalis būtų lygi 1

$-1.11_2 = -1.11 * 2^0$

Rezultate gauname, kad ženklas yra "-", mantisė yra 11, o charakteristika yra 127 (= 0 + 127).

Vaizduojant realų skaičių 4-iuose baituose, ženklui skiriamas 1 bitas (0 reiškia "+", 1 - "-"), mantisei - 23 bitai, o charakteristikai - 8 bitai.

Taigi realus skaičius -1.75 keturiuose baituose būtų pavaizduotas taip:

1	01111111	1100000 00000000 00000000
ženklas	charakteristika	mantisė
(1 bitas)	(8 bitai)	(23 bitai)

Pastaba. Kaip jau buvo minėta, personaliniuose kompiuteriuose jaunesnieji baitai talpinami atmintyje mažesniais adresais, todėl fiziškai atmintyje realus skaičius -1.75 keturiuose baituose bus pavaizduotas ne taip

-1.75 10111111 11100000 00000000 00000000
o šitaip
-1.75 00000000 00000000 11100000 10111111

Kadangi sveikoji skaičiaus dalis turi būti lygi 1 ir ji nėra vaizduojama, tai kyla natūralus klausimas, kaip atmintyje vaizduojamas realus skaičius 0. Iš tikrųjų yra priimtas (realizuotas) susitarimas, kad 0 yra realus skaičius sudarytas iš visų 0-nių bitų, nors formaliai taikant vaizdavimo taisykles (ženklas - 0, mantisė - 0, charakteristika - 0), tai būtų skaičius $+1.0 * 2^{-127}$.

Panagrinėkime, kokios problemos kyla, dirbant su realiais skaičiais.

Beveik visada realūs skaičiai negali būti pavaizduoti tiksliai, tam kad pavaizduoti kompiuterio atmintyje, jie apvalinami, t.y. gaunama apvalinimo paklaida (*angl. roundoff error*). Nors slankaus kablelio formato skaičių kitimo diapazonas labai platus (pavyzdžiui, moduliu maždaug nuo 10^{-45} iki 10^{38}), bet vaizduojant 4-iuose baituose (kai mantisei skiriami 23 bitai), tiksliai pavaizduojami tik 6-7 dešimtainiai skaitmenys. Pastebėkime, kad slankaus kablelio skaičiai išsidėstę labai netolygiai: arti 0 labai tankiai, o kuo toliau nuo 0 tuo rečiau - todėl absoliuti vaizdavimo (apvalinimo) paklaida priklauso nuo skaičiaus (jo modulio) dydžio.

Jei, atliekant operaciją, gaunamas per didelis realus skaičius, kurio nebegalima pavaizduoti atmintyje (faktiškai gaunama per didelė eilė), įvyksta perpildymas (*angl. overflow error*).

Kita galima problema yra reikšmės pametimas (*angl. underflow error*). Jei norime, pavyzdžiui, sudėti du realius skaičius, tai pirmiausia reikia suvienodinti jų eiles. Šio veiksmo rezultate, jei skaičių eilės labai skyrėsi, vienas iš skaičių, kuris buvo nenulinis, gali tapti 0 ir jo pridėjimas nieko neduoda. Tai svarbu žinoti, rašant programas. Pavyzdžiui, reikia parašyti programą, kuri skaičiuoja sumą $\sum 1 / (n^2)$. Jei sumos skaičiavimą suprogramuosime "normaliai", t.y.

$1 + 1 / 2^2 + 1 / 3^2 + 1 / 4^2 + \ldots + 1 / (n^2)$

ir rezultatui naudosime realius skaičius, vaizduojamus 4 baituose, tai skaičiuoti šią sumą, kai n yra didesnis už 5000 nėra prasmės, nes rezultatas nesikeičia, kadangi pridedami nariai pasidaro per maži. Reiktų programuoti tokios sumos skaičiavimą pradedant nuo paskutinio nario, t.y. $1 / (n^2) + \ldots + 1 / 4^2 + 1 / 3^2 + 1 / 2^2 + 1$.

4. OS Branduolys

Branduolys paslepia aparatūrinę įrangą nuo abstraktaus, aukšto lygio programų sluoksnio. Jis atlieka daug darbų, kurių rezultatais naudojasi vartotojo lygmens programos. Pavyzdžiui, branduolys realizuoja ir tvarko šiuos žemiau pateiktus UNIX architektūrinius sprendimus:

- procesus,
- signalus,
- virtualią atmintį,
- failų sistemą,
- sąsają tarp procesų.

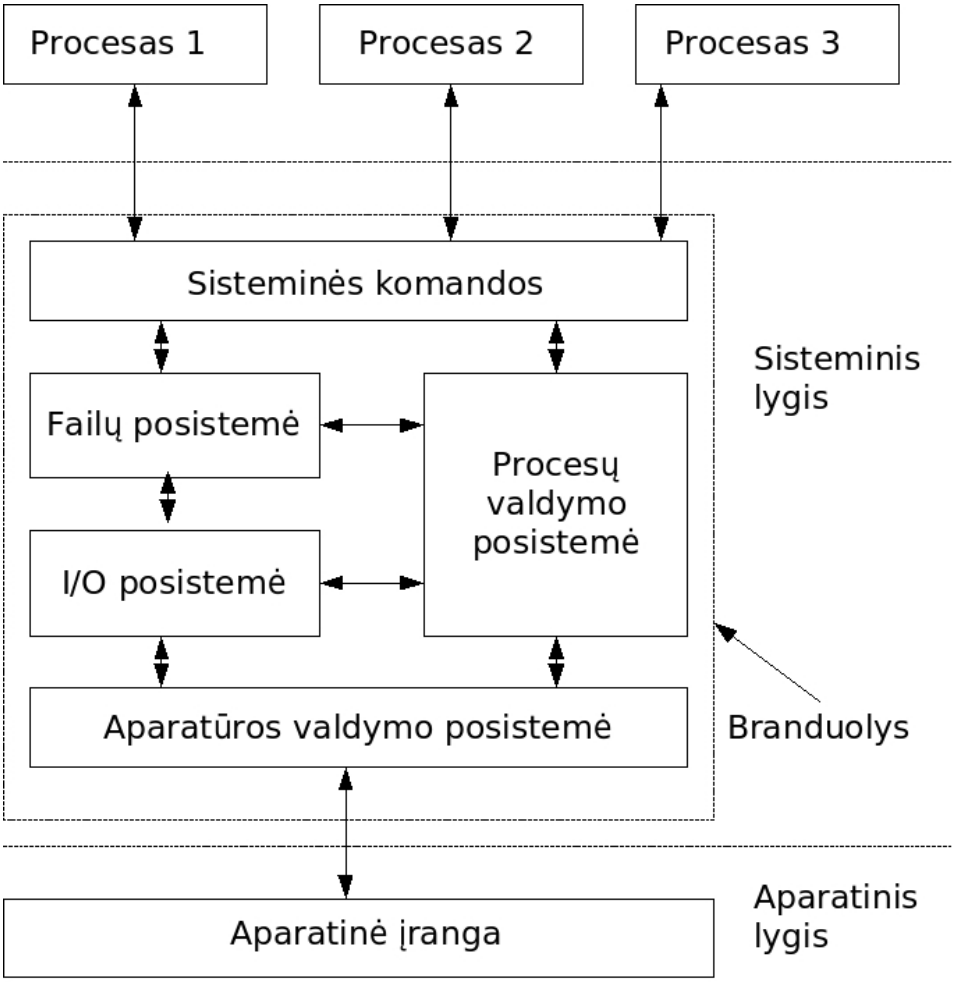
Branduolys tiesiogiai dirba su geležies tvarkyklėmis (device driver), kurie iš dalies ir sudaro branduolio dalį, kurio likusi beveik visa dalis yra nuo geležies nepriklausoma. Branduolys su tvarkyklėmis bendrauja taip pat, kaip ir programos su branduoliu, pavyzdžiui, kai procesas paprašo branduolio “perskaityk pirmus 64 /etc/passwd failo bitus”, tai branduolys nusiunčia tvarkyklei instrukciją, pvz. “Fetch block 3348 from device 3”. Tvarkyklė toliau dirbs savo darbą – suskaldys šią komandą į bitų blokus ir pasiųs juos į aparato valdymo registrus.

Branduolys yra parašytas C kalba su nedideliu kiekiu asemblerio kodo. Pradžioje branduolys buvo mažiukas, gal truputi virš pusės Mb, tačiau dabar, palaikantys sudėtingus tinklo protokolus, daugia-procesiškumą, jie siekia net iki 55MB.

Branduolį sudaro šie trys komponentai:

- procesų ir atminties valdymo posistemė,
- failų posistemė ir
- I/O posistemė.

Bendrą branduolio vidinę struktūrą iliustruoja 1 paveikslas.



1 pav. Bendra OS branduolio struktūra

Failų posistemė

Dauguma duomenų UNIX sistemose yra saugoma failuose, kurie yra išdėstyti hierarchinėje duomenų struktūroje. Failai dažniausiai yra saugomi vietiniame diske, tačiau gali būti ir kituose kompiuteriuose, pvz. NFS (Network File System) leidžia tvarkyti failus nutolusiuose kompiuteriuose. Failai tai pat gali būti saugomi ir kitokiuose kaupikliuose, pvz. disketėse, CD- ROM įrenginiuose ir pan., tačiau mes nagrinėsime paprasčiausius kietuosius diskus.

Pirmoji pasirodė System V failų sistema s5fs, vėliau 4.2BSD sistemose buvo sukurta FFS (Fast File System). Pastaroji yra greitesnė, didesnio funkcionalumo ir patikimumo. Nors šiuolaikinės sistemos naudoja sudėtingesnes failų sistemas, tačiau pagrindinės idėjos išliko.

Failų sistemos hierarchija

Nors UNIX failų sistema nėra labai tvarkinga, tačiau bendrą katalogų ir failų sistemų paskirtį reikia žinoti:

/ (root) failų sistemoje yra / katalogas, branduolys (/unix, /vmunix, /kernel ar /stand), įrenginių failai (/dev), sistemos konfigūracijos failai (/etc), minimali aibė komandų (/sbin,

/bin) ir, kartais, laikinų failų katalogas (/tmp).

/usr – failų sistemoje yra saugomos pagrindinės sistemos programos, man failai ir bibliotekos (/usr/lib, iš kurio yra nuorodos į /lib).

/var – šioje failų sistemoje yra saugomi įvairūs spool, log ir kiti, greitai ir dažnai kintantys failai.

/home – vartotojų namai, kurie paprastai yra saugomi atskiroje failų sistemoje, pvz. /home.

Failų tipai

Paprasti failai.

Tai yra paprasti baitų rinkiniai kurių struktūra OS sistemai yra visai nesvarbi.

Katalogai.

Kataloguose yra saugomos nuorodos į failus. Sakoma, kad failai yra kataloge. Katalogas yra sukuriamas komanda *mkdir*, tuščias pašalinamas *rmdir*, o netuščias – *rm –r*.

Failų vardai kataloge tėra nuorodos į t.t. baitų rinkinį diske – tai yra vadinamasis kietoji jungtis (hard link). Kietąją jungtį galima sukurti su komanda ln, pvz. ln oldfile newfile sukuria dar vieną nuorodą į failą oldfile. Failų atributai, tokie kaip nuosavybė, teisės ir pan. galioja visoms kietoms jungtims.

Simbolinių ir blokinių įrenginių failai.

Tai failai, kurių pagalba yra bendraujama su aparatūrinės įrangos tvarkyklėmis. Šie failai nėra tvarkyklės, jie tik persiunčia komandas tvarkyklėms. Simboliniai aparatūrinės įrangos failai perduoda duomenų srautą nuosekliai, t.y. srauto buferį realizuoja tvarkyklės.

Blokinių failų tvarkyklės dirba su dideliais bitų blokais ir, tokiu atveju, srauto buferius realizuoja branduolys. Aparatūrinės įrangos failai yra sukuriami su komanda *mknod*. Dauguma sistemų suteikia /dev/MAKEDEV programą, kuri sukuria šiuos failus.

Vietiniai kanalai (domain sockets).

Tai yra tarp procesiniai ryšio kanalai. Procesų sukurtus kanalų failus gali skaityti vartotojai, tačiau juos modifikuoti gali tik procesai, kuriuos betarpiškai sieja konkretus kanalas. Šiuos kanalus sukuria socket() sisteminė funkcija, o sunaikina rm (arba unlink()), kai jis tampa nebereikalingas.

FIFO kanalai (Named Pipes).

Kaip ir vietiniai, FIFO kanalai sukuria ryšį tarp procesų, tačiau jie realizuoja FIFO duomenų struktūrą.

Simbolinė (minkšta) nuoroda.

Simbolinė nuoroda, skirtingai nuo kietosios, yra surišama ne tiesiogiai, per adresą, o per vardą. Kai branduolys vykdo operaciją simbolinės nuorodos kataloge, jis iš tikrųjų dirba kataloge, į kurį rodo nuoroda. Simbolinę nuorodą galime sukurti naudojant tiek absoliutų, tiek santykinį kelią.

Pavyzdžiui, jei norime kataloge */usr/include/ufs* turėti katalogo */usr/include/bsd/sys/ufs* turinį, rašome tokią komandą:

 # ln –s ../../ufs /usr/include/bsd/sys/ufs

Failų atributai

Failo atributai nurodo failo savininką (-us), tipą ir pan. Kiekvienas failas turi 16 bitų (vieną žodį) skirtą failo atributams saugoti. 9 leidimo bitai nurodo, kas gali skaityti, modifikuoti ir vykdyti failą. Su sekančiais 3-mis bitais, kurie valdo failo vykdymą, jie sudaro failo būseną (mode). Likę 4 bitai nurodo failo tipą, kuris yra nustatomas failą sukuriant ir vėliau nebekeičiamas. Kitus 12 bitų gali keisti failo savininkas arba super-vartotojas (root) naudodami komandą *chmod*.

Failai taip pat turi *setuid* ir *setgid* bitus, kurie nurodo failo savininkus –vartotoją ir grupę. Naujai sukurti failai įgauna katalogo, o ne jį sukūrusio vartotojo *setgid* reikšmę. Taip yra palengvinama apsikeitimo failais funkcija.

Taip pat kartais dar yra naudojamas „kibusis“ (sticky) bitas. Jo paskirtis PDP mašinose, kuriose, beje, UNIX pradėjo savo dienas, buvo požymis, kad failas turi likti atmintyje ar swap įrenginyje. Šiuo metu, kai atmintis yra labai pigi, kibusis bitas yra ignoruojamas, tačiau jis vis dar lieka svarbus operuojant failais – ne failo savininkas ar super-vartotojas negali failo ištrinti, nors tokias teises ir turi.

9 leidimų bitai yra organizuoti sekančiai: 400, 200 ir 100 yra failo savininko, 40, 20 ir 10 – grupės, 4, 2 ir 1 – visų kitų vartotojų. Didžiausios reikšmės (400, 40 ir 4) nurodo leidimą skaityti failą, viduriniosios (200, 20 ir 2) – rašymo ir mažiausios (100, 10 ir 1) – vykdymo. Failo trynimą ir pervadinimą valdo katalogo, kuriame betarpiškai yra failas, leidimo bitai.

5. Procesų valdymo posistemė

UNIX OS širdis – procesų valdymo posistemė. Visas OS funkcionalumas galų gale priklauso nuo vienų ar kitų procesų vykdymo. Viskas servisai, spausdinimas ir pan. yra t.t. procesų vykdymo pasekmė. Procesai UNIX sistemoje yra tiesiogiai susiję su dviem resursais:

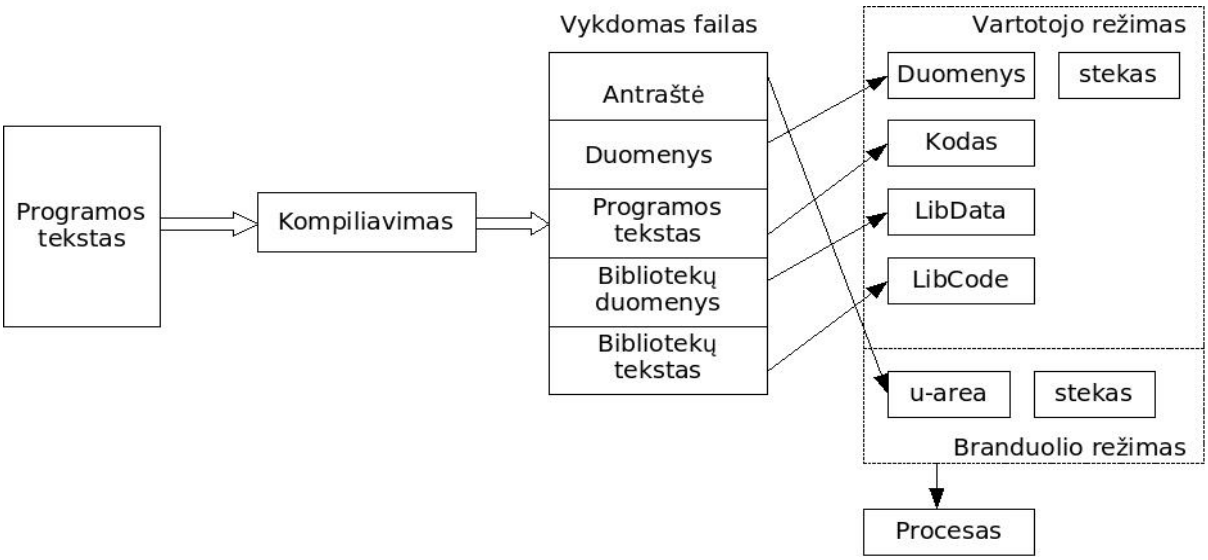
procesoriumi (-iais) ir operatyvine atmintimi. Galioja taisyklė, kad šių resursų niekada nėra per daug, todėl OS pastoviai vyksta arši konkurencinė kova dėl jų.

Nagrinėsime, kokiu būdu OS branduolys valdo atmintį, kai kiekvieno proceso adresinė erdvė siekia kelis gigabaitus, kaip OS vienu metu gali vykdyti daug procesų, kai vienas CPU tegali vykdyti tik vieną užduotį.

Procesas, tai rėmelis į kurį yra įstatoma programa. Nors viena iš branduolio užduočių yra proceso izoliavimas, yra pakankamai svarbus ir duomenų tarp procesų apsikeitimo organizavimas. Tam UNIX suteikia nemažai priemonių – nuo paprastų signalų iki sudėtingų tarp-procesinės sąveikos mechanizmų – IPC SV ir soketų BSD sistemose.

Procesų valdymo pagrindai

Procesas, tai vykdomos programos pavidalas, tam tikros branduolio duomenų struktūros su nuorodomis į atmintyje esančius vykdomą kodą, duomenis, steką ir bibliotekas (2 pav.).



2 pav. Bendroji proceso struktūra

Pradžioje UNIX vienu metu galėjo vykdyti tik du procesus, t.y. po vieną kiekvienam prie PDP-7 prijungtiems terminalams. Vėliau šis procesų skaičius gerokai padidėjo, atsirado sisteminė komanda (system call) fork(2). System 1 jau turėjo exec(2) komandą, bet vienu metu galėjo pakrauti tik vieną procesą. Po to, kai į PDP-11 buvo įdiegtas MMU (Memory Management Unit) tapo įmanoma operatyvinėje atmintyje laikyti kelis procesus, taip sumažinant I/O operacijas. Tačiau iki pat 1972, kol UNIX nebuvo perrašytas C kalba, visos I/O operacijos buvo sinchroniškos, t.y. kol vienas procesas nebaigdavo I/O operacijos, visi kiti privalėjo laukti. Nuo 1973 metų procesų valdymo pagrindai praktiškai nepasikeitė. Proceso vykdymas CPU vyksta dviejuose režimuose:

1. vartotojo režimas (user mode) yra vykdomos programos instrukcijos. Tuo metu procesoriui nėra prieinami sisteminiai resursai. Kai jam prireikia kokio nors sisteminio resurso, tarkim skaitymo iš failo, jis atlieka sisteminę komandą ir pereina į branduolio režimą.

2. Branduolio režimas (kernel mode). Jame yra vykdomos sisteminės komandos, apdorojamas sisteminis iškvietimas. Tokiu būdu vartotojo programa yra izoliuojama nuo sisteminių duomenų struktūrų (negali pakenkti) ir t.t. komandos tegali būti vykdomos branduolio režime, tarkim registro reikšmių keitimas.

Dėl šios proceso dvilypybės, proceso atvaizdas atmintyje yra iš dviejų dalių – užduoties ir branduolio duomenų. CPU dirbdamas branduolio režimu turi priėjimą prie visų duomenų aplamai.

Viena pagrindinių OS funkcijų – operatyvinės atminties (RAM) valdymas, nes operatyvinė atmintis yra brangus resursas. Duomenys iš procesoriaus į atmintį keliauja tik kelis CPU taktus, t.y. labai greitai, todėl ten laikyti duomenis yra labai efektyvu. Kadangi pagrindinė atmintis yra ribota, netelpantys duomenys yra saugomi antrinėje atmintyje (swap), kurios vaidmenį dažniausiai atlieka diskas. Diskai yra gerokai lėtesni ir reikalauja papildomos OS priežiūros.

UNIX atlieka efektyvų operatyvinės atminties paskirstymą tarp procesų, pirminės ir antrinės atminties valdymą. Dalį šito darbo atlieka atminties valdymo modulis – MMU (Memory Management Unit).

Reikalavimai operatyvinės atminties valdymo moduliui:

- uždavinių, didesnių nei atminties kiekis, vykdymas;
- dalinai pakrautų uždavinių vykdymas;
- kelių uždavinių pakrovimas ir efektyvus vykdymas;
- uždavinio pakrovimas į tam tikrą vykdymo sritį;
- uždavinio suskaidymas ir saugojimas įvairiose priminės ir antrinės atminties srityse;
- užtikrinti bendrą atminties sričių naudojimą keliems uždaviniams.

Visa tai realizuojama UNIX OS su virtualios atminties pagalba. Tačiau tai kainuoja: virtualios atminties valdymo mechanizmas užima atmintį, reikalauja procesoriaus ir ilgai trunkančių I/O operacijų. Vidutinio apkrovimo sistemose apie 7% procesoriaus laiko užima virtualios atminties valdymo mechanizmo realizacija. Todėl nuo efektyvaus jo darbo priklauso ir visos OS efektyvumas.

Šiuolaikinėse sistemose kiekvienas procesas dirba savo virtualios atminties erdvėje ir jam susidaro įspūdis, kad visa atmintis priklauso tik jam vienam ir jos yra tikrai pakankamai. Procesai tampa izoliuoti vienas nuo kito.

6. Procesai

Proceso sąvoka

Programos tekstas yra įvairių instrukcijų seka, o jos vykdymas – dinamiškas procesas. Programa yra pasyvi, kol saugoma išorinėje atmintyje, ir tada ji neturi jokios įtakos kompiuteriui. Programa tampa aktyvi, kai įrašoma į operatyviąją atmintį ir OS ją pradeda vykdyti.

Aktyvi programos būsena vadinama procesu. Taigi procesas yra tam tikras darbas, atliekamas pagal programą. Procesą kiekvienu laiko momentu apibūdina jo būsena: jis gali būti vykdomas, laukiantis, paruoštas vykdyti. Atskiras proceso būsenas, perėjimą iš vienos į kitą, taip pat nuo vieno proceso prie kito valdo OS. **Beveik visais atvejais procesą galima traktuoti, kaip procesoriaus darbą, apdorojant programą su jos duomenimis.**

Paprastai sistemoje vienu metu vyksta daug procesų, kurių dalis gali būti OS procesai (vykdo sistemines užduotis), o kiti – vartotojo procesai (vykdo vartotojo taikomąją programą). Kai kuriems procesams gali prireikti tų pačių kompiuterio resursų, todėl OS turi juos tinkamai paskirstyti.

Valdydama procesus, OS atlieka tokias pagrindines funkcijas:

- sukuria ir baigia vartotojo ar sisteminį procesą;
- sustabdo ir vėl atkuria procesus,
- sinchronizuoja procesus;
- palaiko ryšius tarp procesų.

Šiuolaikiniai kompiuteriai gali atlikti keletą užduočių vienu metu. Daugiaužduotiškumo atveju sistemos procesorius pereina nuo vienos užduoties prie kitos, skirdamas kiekvienai užduočiai 10 ar 100 milisekundžių. Todėl vartotojui susidaro įspūdis, kad programos vykdomos lygiagrečiai. Nors kiekvienu konkrečiu laiko momentu procesorius atlieka tik vieną užduotį.

Sistemoje turinčioje vieną procesorių kiekvienu laiko momentu yra atliekamas vienas procesas. Kiti procesai laukia savo eilės. Kiekvienas naujas procesas atsirandantis sistemoje pereina į būsena, kuri vadinama paruoštas. Operacinė sistema savo ruoštu naudoja tam tikrą planavimo algoritmą ir vieną paruoštų procesų perkelia į vykdymo būsena. Išeiti iš šios būsenos procesas gali dėl šių priežasčių:

· OS nutraukia proceso vykdymą.

· Procesas negali toliau tęstis, nes trūksta duomenų, arba neįvyko tam tikras įvykis.

Tuomet OS perveda procesą į laukimo būseną.

· Įvykus pertaukimui.

Taigi savo gyvavimo ciklo metu procesas pereina iš vienos būsenos į kitą priklausomai nuo OS realizuotų procesų valdymo algoritmų. Žemiau pateikiami būsenų apibūdinimai:

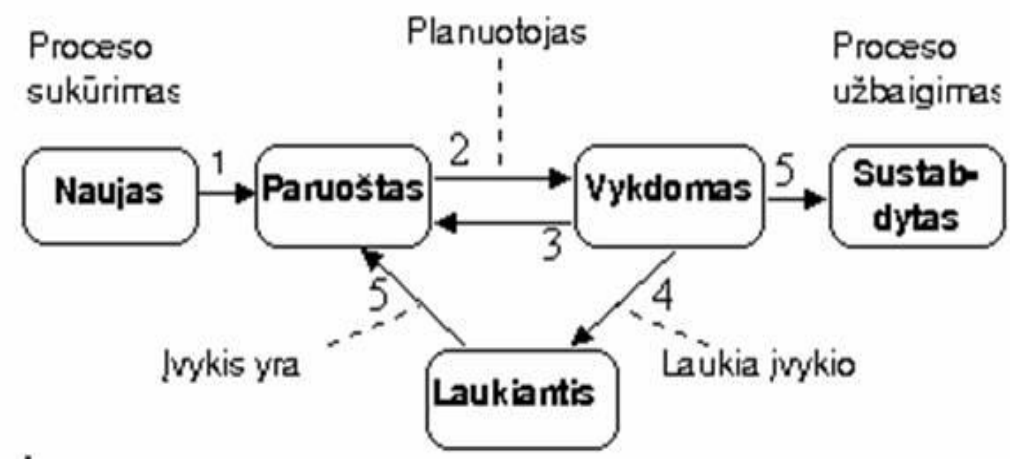
Naujas procesas - ką tik sukurtas - turi savo proceso ID.

Paruoštas procesas - laukia, kol atsilaisvins procesorius; multiprogramin̄eje sistemoje gali būti daug tokių procesų.

Vykdomas procesas - vienintelis, jei sistema vienaprocesorinė; multiprocesorinėje sistemoje tokių procesų gali būti daug.

Laukiantis (blokuotas) procesas - buvo vykdomas, bet dabar laukia kokio nors įvykio (pvz., duomenų įvedimo ar išvedimo).

Sustabdytas (užbaigtas) procesas - buvo baigtas normaliai ar nutrauktas dėl kokių nors priežasčių; sistema turi registruoti šį faktą.



3 pav. Procesų būsenos

Vienaprocesorinėje sistemoje vykdymo būsenoje gali būti tik vienas procesas, o paruošto ir laukimo būsenose – keletas procesų, šie procesai sudaro atitinkamai laukiančiuosius ir pasiruošusius procesus. Proceso gyvavimo ciklas prasideda nuo paruošto būsenos, kada procesas yra pasiruošęs vykdymui ir laukia savo eilės. Procesas negali savarankiškai pereiti iš vienos būsenos į kitą, tai atlieka operacinė sistema.

Procesų deskriptorius

Tam kad operacinė sistema galėtų valdyti procesus, OS reikalinga papildoma informacija, tai:

- proceso būsena,
- programinio skaitiklio parodymai, t.y. komandos, kuri turi būti atlikta adresas,
- procesoriaus registrų turinys,
- procesų planavimui reikalingi duomenys: proceso identifikatorius, duomenys apie proceso privilegijas, dydis, kokia proceso trukmė ir koks vartotojas pradėjo procesą,
- duomenys apie įvedimo/išvedimo įrenginius, susiję su procesu.

Kiekvieno proceso duomenų struktūra priklauso ir nuo operacinės sistemos. Dažniausiai visa informacija apie procesus saugoma keliose duomenų struktūrose. Procesų planavimui ir valdymui reikalinga informaciją, proceso identifikatorius, duomenys apie proceso privilegijas, kodo segmento vieta ir kita - vadinama proceso deskriptoriumi. O proceso kodas ir duomenys vadinami proceso kontekstu. Bet kuriuo laiko momentu procesą pilnai charakterizuoja, kontekstas.

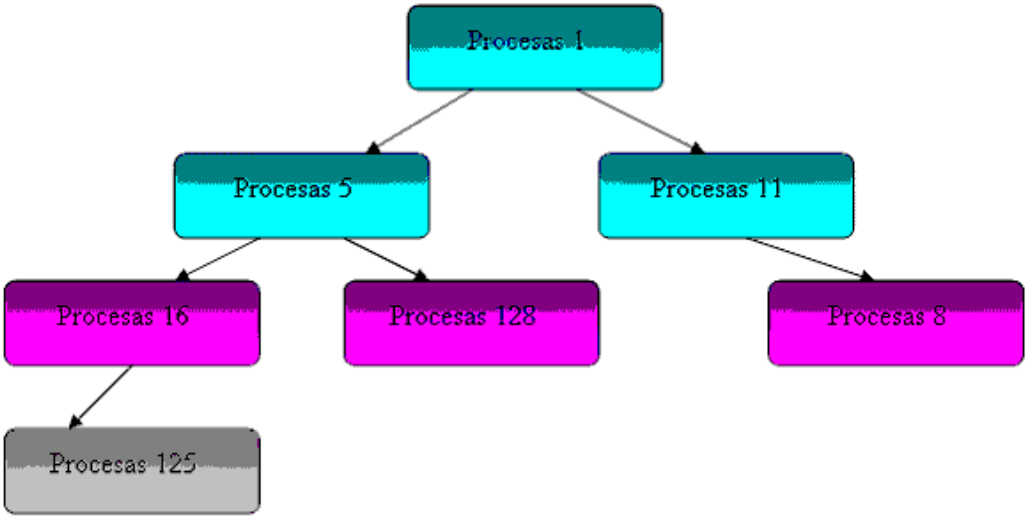
Procesų kūrimas

Operacinė sistema palaikanti proceso koncepciją, turi turėti priemones naujo proceso sukūrimui. Labai paprastose sistemose, visi procesai gali būti sukurti sistemos pasikrovimo eigoje. Sudėtingesnėse os procesai kuriami dinamiškai iškilus būtinybei.

Naujam procesui:

1. Priskiriamas identifikacinis numeris. Šiame etape procesų lentelėje patalpinamas naujas įrašas.
2. Išskiriama atminties vieta procesui. OS turi žinoti, kiek vietos reikia vartotojo adresų sričiai (programai ir duomenims) ir vartotojo stekui.
3. Proceso valdančiojo bloko inicializacija.
4. Ryšių nustatymas.
5. Naujų duomenų struktūrų sukūrimas arba esamų papildymas.

Paprastai kraunantis os yra sukuriami keli procesai. Kiekvienas procesas gali sukurti vieną ar kelis naujus procesus. Procesas, kuris sukuria naują procesą vadinamas procesu - tėvu, o naujai sukurtas vadnamas procesu– vaiku. Procesai – vaikai gali sukurti naujus procesus – vaikus, tokiu būdu formuodami geneologinį medį Pav. 4.



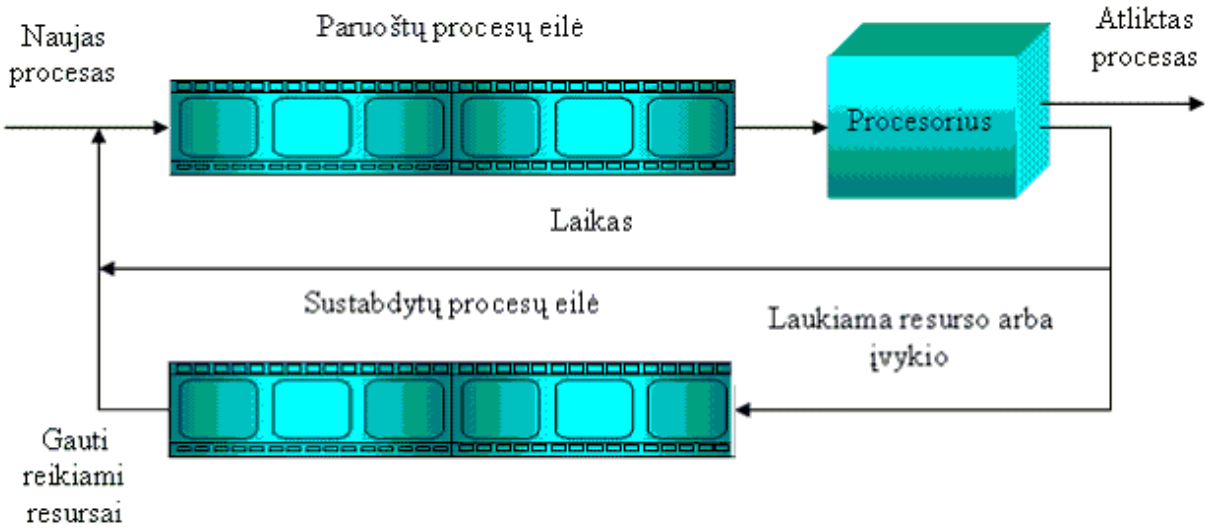
4 pav. Procesų geneologinis medis.

Mums įdomu, kaip elgiasi procesas – tėvas turintis keletą procesų – vaikų. Procesas – tėvas gali tęsti savo darbą vienu metu su procesu – vaiku, arba gali laukti kol baigs darbą procesas – vaikas arba vaikai.

Procesų eilės

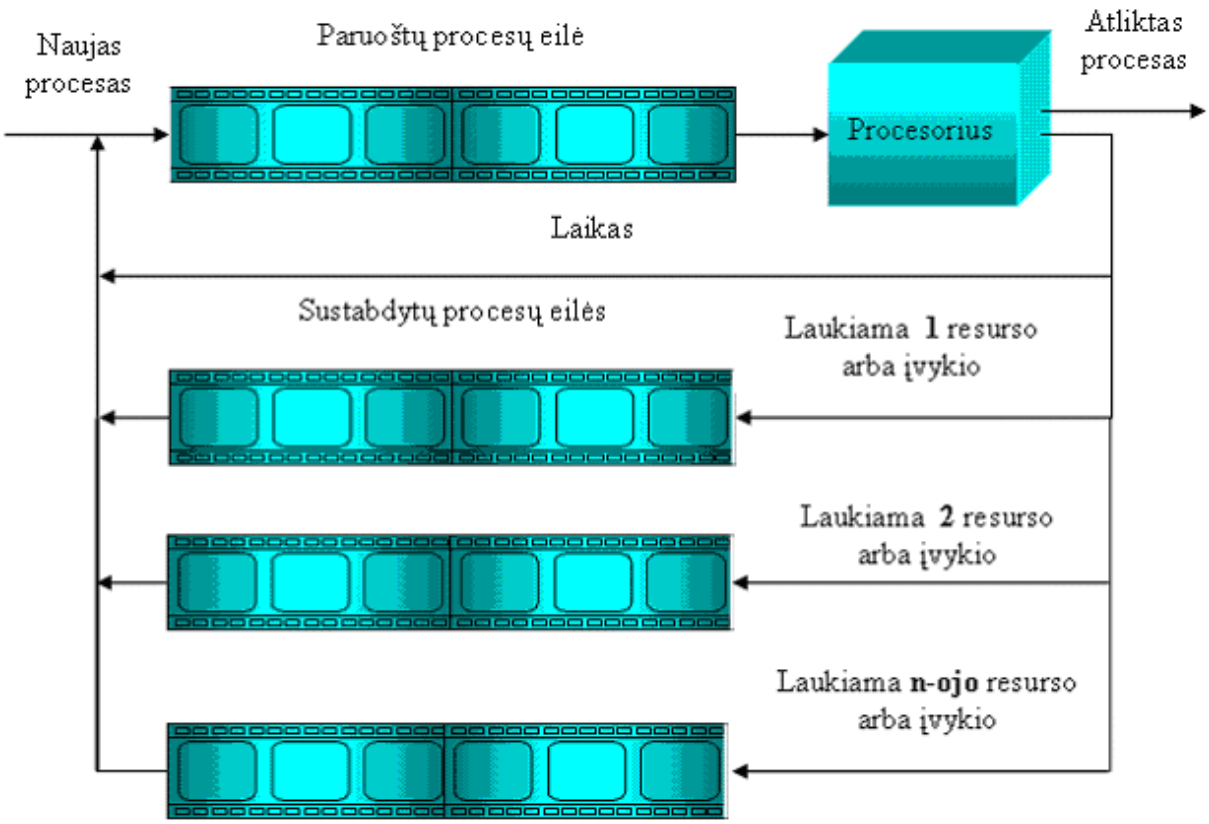
Atrodytų, kad pati persijungimo funkcija tarp procesų yra suprantama ir paprasta. Tam tikru laiko momentu nutraukiamas vykdomas procesas ir pereinama prie kito proceso vykdymo. Tačiau kyla klausimas, kokie įvykiai įtakoja persijungimą nuo vieno proceso prie kito, kaip reikia os elgtis su skirtingom duomenų struktūromis.

Panagrinėkime Paveikslą Nr. 5. Jame pateiktas procesų eilių modelis, su viena sustabdytų procesų eile. Pagal šį modelį yra dvi procesų eilės: viena paruoštiems procesams, kita sustabdytiems.



5 pav. Procesų eilių modelis, su viena sustabdytų procesų eile

Visi naujai sukuriami procesai patenka į paruoštųjų eilę. Kada operacinei sistemai reikia pasirinkti procesą vykdymui, ji pasirenka procesą iš paruoštų eilės. Jei neatsižvelgiama į prioritetus atranka vyksta pagal principą: pirmas patekai – pirmas vykdomas. Jeigu proceso vykdymas yra nutraukiamas, jis priklausomai nuo sąlygų gali patekti tiek į paruoštų, tiek į sustabdytųjų eilę. Ir galiausiai, jeigu įvyksta įvykis, kurio laukia sustabdyti procesai, visi jie perkeliami į paruoštųjų eilę. Dėl tokios organizacijos operacinė sistema turi tikrinti sustabdytų procesų eilę, vos tik įvyksta bet koks įvykis, tam kad rastu procesus, kurie laukė būtent šio įvykio. Todėl efektyviau būtų organizuoti kelias eiles, kiekvienam įvykiui savo. Tuomet įvykus tam tikram įvykiui bus galima perkelti visus atitinkamos eilės procesus perkelti į paruoštų eilę (6 pav.).



6 pav. Procesų eilių modelis, su keliomis sustabdytų procesų eilėmis

Kada reikia perjungti procesus?

Persijungimas tarp procesų gali įvykti, bet kuriuo momentu, kai vykdomo proceso valdymas pereina operacinei sistemai. Žemiau yra pateikiamos galimos priežastys, dėl kurių valdymas yra perduodamas operacinei sistemai.

- Pertraukimas
- Spąstai
- Supervizoriaus iškvietimas

Pirmiausiai apžvelgsime sisteminius pertraukimus. Faktiškai yra išskiriami du sisteminių pertraukimų tipai: įprastiniai pertraukimai ir spąstai (trap). Įprastiniai pertraukimai įvyksta dėl įvykių, kurie nėra susiję su vykdomu procesu, ir yra išoriniai jo atžvilgiu (pav. įvedimo/išvedimo operacijos pabaiga). Spąstai yra tiesiogiai susiję su vykdomo proceso klaidom. Pav. tai galėtų būti bandymas neteisėtai gauti prieigą prie failų. Štai keletas pertraukimo pavyzdžių:

Taimerio petraukimai. Operacinė sistema nustato ar proceso vykdymo laikas yra lygus maksimaliam leistinam laiko tarpui. Jeigu tai tiesa, vykdomo proceso duomenys yra perkeliami į pasiruošimo būseną ir vykdomas kitas procesas.

Įvedimo/išvedimo pertraukimai. Proceso vykdymo metu paspaudžiamas bet koks klaviatūros mygtukas, tuomet įvyksta pertraukimas ir proceso vykdymas nutraukiamas.

Pertraukimai dėl resursų stokos. Proceso vykdymo metu paaiškėja, kad trūksta kokio nors resurso, pav. duomenų, tuomet proceso vykdymas nutraukiamas. Vykdomo proceso duomenys yra perkeliami į pasiruošimo būseną ir vykdomas kitas procesas.

Spąstų atveju operacinė sistema sprendžia ar įvykusi klaida yra fatalinė, jeigu taip procesas nustojamas vykdyti ir gražinamas, kaip baigtas procesas. Priklausomai nuo operacinės sistemos gali būti vartotojui parodomas sisteminis klaidos pranešimas, arba pati os gali pereiti prie kito proceso arba toliau bandyti vykdyti jau pradėtą.

Kritinė sekcija

Kai kuriose OS keli procesai gali naudotis bendrais duomenimis, juos skaityti, papildyti. Panagrinėsime labai paprastą pavyzdį, kuris puikiai parodo iškylančias problemas, kai naudojami bendri resursai.

Sakykime procesui reikia atspausdinti failą, jis patalpina failo vardą į specialų spulerio katalogą. Kitas procesas, atsakingas už spausdinimą, periodiškai tikriną ar yra failų, kuriuos reikia spausdinti.

Suradęs - spausdina failą ir pašalina jo vardą iš katalogo.

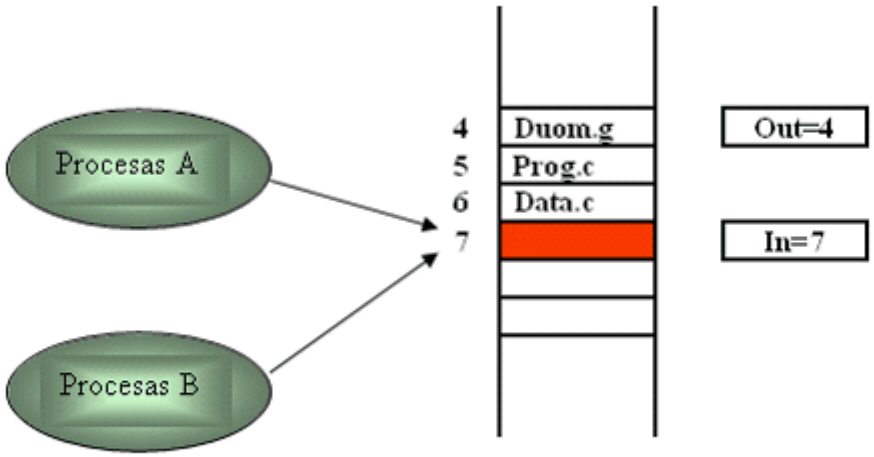
Įsivaizduokite, kad spulerio katalogą sudaro didelis skaičius segmentų, sunumeruotų 0, 1, 2, 3 ir t.t, kurių kiekviename gali būti saugomas failo vardas (Pav. 5). Taip pat yra du kintamieji: **out**, rodo į kitą spausdinimui skirtą failą ir **in**, rodantį į kitą tuščią segmentą. Šiuos du kintamuosius galima saugoti viename faile, prieinamame visiems procesams.

Sakykime, kad esamuoju laiko momentu segmentai nuo 0 iki 3 yra tušti (šie failai jau atspausdinti), o segmentai nuo 4 iki 6 yra užimti (šie failai laukia savo eilės, kad būtų atspausdinti). Tuo pačiu metu procesai A ir B nusprendžia savo failus nusiųsti spausdinimui.

Pagal Merfio dėsnį galima tokia situacija. Procesas A perskaito in kintamojo reikšmę (7) ir išsaugo savo lokaliame kintamajame, kaip tuščią segmento adresą. Tuo metu įvyksta taimerio pertraukimas ir procesorius persijungia prie Proceso B. Procesas B perskaito in kintamojo reikšmę (ji vėl yra 7) ir išsaugo savo lokaliame kintamajame, kaip tuščią segmento adresą. Duotu momentu abu procesai mano, kad kitas laisvas segmentas yra 7.

Procesas B spulerio kataloge išsaugo failo vardą ir pakeičia in reikšmę į 8, bei atlieka kitus veiksmus nesusijusius su spausdinimu. Pagaliau valdymas pereina Procesui A ir jis tęsia darbą nuo ten kur baigė. Nuskaito, savo lokaliam kintamajame tuščio segmento adresą 7 ir įrašo failo vardą(aišku ištrindamas ten Proceso B įrašytą failą). Spulerio katalogo struktūra nepažeista, todėl procesas atsakingas už spausdinimą neko neįtars , tik Proceso B failas nebus

atspausdintas. Tokioje situacijoje laimi tas procesas, kuris buvo pirmas. Kyla klausimas kaip išvengti lenktynių efekto tam tikro resurso atžvilgiu?



7 pav. Procesų spooler‘is

Suformuluokime lenktynių būseną abstrakčiame lygmenyje. **Programos dalis, kurioje naudojamasi bendrais duomenimis yra vadinama kritine sekcija.** Kad išvengti lenktynių efekto tam tikro resurso atžvilgiu, reikia užtikrinti, kad kiekvienu momentu kritinėje sekcijoje, susietoje su šiuo resursu, būtų tik vienas procesas. Toliau bus nagrinėjami būdai kaip nepatekti į kritinę sekciją, jeigu ten jau yra kitas procesas.

Problemos sprendimo būdai:

Uždrausti visus pertraukimus.

Kai tik procesas patenka į kritinę sekciją, reikėtų uždrausti visus pertraukimus. Kadangi procesorius persijungia nuo vieno proceso prie kito tik pertraukimų dėka, tuomet procesas, esantis kritinėje sekcijoje, galės ramiai skaityti ir įrašyti duomenis, nes joks kitas procesas jam nesutrukdys. Tačiau šis problemos sprendimo būdas yra netinkamas, nes pavojinga patikėti sistemos valdymą vartotojo procesui; jis gali ilgam užimti procesorių, o „pakibus“ procesui kritinėje sekcijoje, krachą patirs visa sistema, kadangi pertraukimai niekada nebus įvykdyti.

Blokuojančių kintamųjų panaudojimas.

Su kiekvienu bendru resursu susiejamas dvejetainis kintamasis, kuris įgyja reikšmę 1, jeigu resursas laisvas (t.y. nei vienas procesas nėra šiuo momentu kritinėje sekcijoje, susijusioje su šiuo resursu), ir reikšmę 0, jei resursas užimtas. Prieš įeidamas į kritinę sekciją, procesas patikrina kintamojo reikšmę. Jei kintamojo reikšmė yra 1, tai tikrinimas ciklišKai tęsiamas, jei 0, tai procesas įeina į kritinę sekciją.

Tačiau ir šis problemos sprendimo būdas turi trūkumų. Problema ta pati, kaip ir su spulerio katalogu. Įsivaizduokime, kad vienas procesas nuskaito kintamojo reikšmę ir ji yra lygi 0, bet jam dar nesuspėjus reikšmės pakeisti į 1, valdymas perduodamas kitam procesui, kuris sėkmingai pakeičia kintamojo reikšmę į 1. Kai pirmam procesui bus perduotas valdymas, jis pakeis kintamojo reikšmę į 1 ir abu procesai atsidurs kritinėje sekcijoje.

Griežtas eiliškumas.

Kaip ir aukščiau aptartu atveju, bus naudojamas vienas kintamasis, kurio pradinė reikšmė yra lygi 0. Tik ši reikšmė bus naudojama tam, kad parodyti, kuris procesas gali įeiti į kritinę sekciją duotu momentu. T. y. procesai į kritinę sekciją įeina griežtai pagal eilę: P0, P1, P0, P1… Procesas gali patekti į kritinę sekciją tik tuomet, kai kintamojo reikšmė yra lygi proceso eilės numeriui. Šis metodas visiškai netinka, kai procesu trukmė labai skiriasi. Pav. procesas0 baigė darbą kritinėje sekcijoje ir kintamojo reikšmė pasidarė lygi 1, o procesas1 dar nebaigė savo darbo ne kritinėje sekcijoje. Šiuo atveju abu procesai yra nekritinėje sekcijoje. Netikėtai procesas0 baigė darbą ir nori pereiti I kritine sekcija, bet negali nes kintamojo reikšmė yra lygi 1, o kritinė sekcija laisva.

Semaforų primityvai.

Apibendrintą procesų sinchronizavimo būdą pasiūlė Deikstra. Deikstra (1965 m.) įvedė du naujus primityvus, kurie žymiai suprastino procesų tarpusavio ryšius ir sinchronizaciją. Abstrakčioje formoje šie primityvai, žymimi P ir V, operuoja neneigiamais sveikaisiais kintamaisiais, vadinamais semaforais. Tegul S – toks semaforas. Operacijos apibrėžiamos tokiu būdu:

- Operacija V(S): kintamasis S didinamas vienetu (inkrementas) vienu nedalijamu veiksmu; atrinkimas, inkrementas ir įsiminimas negali būti pertraukiami, ir S negalima pasiekti kitu procesu operacijos metu.
- Operacija P(S): jei į manoma, S sumažinamas vienetu. Jeigu S=0, tai neįmanoma S sumažinti ir likti sveikųjų neneigiamų reikšmių srityje; tada procesas, iškviečiantis P-operaciją, laukia, kada bus galima sumažinti. Sėkmingas patikrinimas ir S sumažinimas – taip pat nedalijama operacija.

Jeigu keletas procesų vienu metu reikalauja P- ar V-operacijos tam pačiam semaforui, tai tos operacijos bus vykdomos iš eilės laisva tvarka; analogiškai, jeigu daugiau negu vienas procesas laukia P-operacijos vykdymo ir keičiamas semaforas tampa teigiamu, tai konkretus laukiantis procesas, kuris išrenkamas pabaigti operaciją, yra laisvas ir nežinomas.

Būtent semaforiniai kintamieji naudojami procesų sinchronizavimui. P-primityvas turi savyje potencialų laukimą skubių procesų, tuo tarpu V-primityvas gali, galbūt, aktyvizuoti kai kuriuos laukiančiuosius procesus. P ir V nedalomumas garantuoja, jog semaforo reikšmė bus sveikas skaičius.

Aklavietės

Pateikto pavyzdžio pagalba panagrinėsime dar vieną sinchronizavimo problemą – tarpusavio blokavimą, vadinamą aklavietėmis (deadlocks, clinch). Jeigu operacijas P(e) ir P(b) rašymo programoje sukeisti vietomis, tai tam tikromis aplinkybėmis šie du procesai gali blokuoti vienas kitą. Iš tikrųjų, tegul „rašytojas“ pirmas įeis į kritinę sekciją ir nustatys, kad nėra laisvų buferių; jis lauks, kol „skaitytojas“ paims eilinį įrašą iš buferio, bet „skaitytojas“ negalės to padaryti, nes negalės įeiti į kritinę sekciją, įėjimas į kurią užblokuotas proceso „rašytojas“. Panagrinėkime dar vieną aklavietės pavyzdį.

Sakykime, dviem procesams, vykdomiems multiprogramiame režime, jų darbo atlikimui reikalingi du resursai, pvz., spausdintuvas ir diskas. Tegul po to, kai procesas A užėmė spausdintuvą (nustatė blokuojantįjį kintamąjį), jis buvo pertrauktas. Valdymą gavo procesas B, kuris iš pradžių užėmė diską, bet vykdant kitą komandą buvo užblokuotas, kadangi spausdintuvas ja užimtas proceso A. Valdymą vėl gauna procesas A, kuris bando užimti diską ir yra blokuojamas: diskas jau yra užimtas proceso B. Tokioje būsenoje procesai A ir B gali būti be galo ilgai. Priklausomai nuo procesų greičio santykio, jie gali arba visiškai nepriklausomai naudoti bendrus resursus (d), arba sudaryti eiles prie bendrų resursų (c), arba blokuoti vienas kitą (b).

Aklavietes reikia skirti nuo paprastų eilių, nors ir vieni, ir kiti atsiranda bendrai naudojant resursus ir išoriškai panašūs: procesas sustabdomas ir laukia resursų atlaisvinimo. Viena eilė – tai normalus reiškinys, rodantis, aukštą resursų išnaudojimo efektyvumą, esant atsitiktinėms užklausoms. Eilė atsiranda tada, kai resursas duotuoju momentu užimtas, bet po kurio laiko jis atsilavina ir procesas tęsia savo vykdymą. Aklavietės, gi, kaip matyti ir iš pavadinimo, yra tam tikru laipsniu neišsprendžiama situacija. Išnagrinėtuose pavyzdžiuose aklavietę sukūrė du procesai, bet vienas kitą blokuoti gali ir daugiau procesų. Aklaviečių problema jungia šiuos uždavinius:

- Aklaviečių išvengimas,
- Aklaviečių atpažinimas,
- Sistemos atstatymas po aklaviečių.

Aklaviečių galima išvengti rašant programą, t.y. programos turi būti taip parašytos, kad aklavietės negalėtų atsirasti prie jokių procesų tarpusavio greičių santykio. Išnagrinėtame pavyzdyje, jeigu procesai A ir B pareikalautų resursų ta pačia seka, tai aklavietės nebūtų galimo iš principo. Kitas aklaviečių išvengimo būdas vadinamas dinaminiu ir jo esmė priskirti resursus procesams, naudojantis tam tikromis taisyklėmis, pvz., resursai gali būti išskiriami tam tikra tvarka, bendra visiems procesams.

Kai kuriais atvejais, kai aklavietės situacija sudaroma daugelio procesų, naudojančių daug resursų, aklavietės atpažinimas yra netrivialus uždavinys.

Egzistuoja formalūs, programiškai realizuoti aklaviečių atpažinimo metodai, pagrįsti bendrų resursų lentelių ir užklausų į užimtus resursus lentelių naudojimu. Analizuojant šias lenteles, galima aptikti tarpusavio blokavimus. Jeigu atsirado aklavietė, tai nebūtina nutraukti visų užblokuotų procesų vykdymą. Galima nutraukti tik dalį iš jų ir atsilaisvins resursai, laukiami kitų procesų. Galima grąžinti kai kuriuos procesus į svopingo sritį. Galima „grąžinti“ į tam tikrus kontrolinius taškus, kuriuose įsimenama visa informacija, būtina programos vykdymo nuo šios vietos atnaujinimui. Kontroliniai taškai išdėliojami programoje tose vietose, po kurių gali atsirasti aklavietės. Taigi, semaforus reikia naudoti atsargiai, kadangi net ir nedidelė klaida gali sustabdyti sistemą.

Kad palengvinti korektiškų programų rašymą, buvo sukurta aukšto lygio sinchronizavimo priemonė – monitorius.

Monitorius – tai procedūrų, kintamųjų ir duomenų struktūrų rinkinys. Procesai gali iškviesti monitoriaus procedūras, bet negali naudotis vidiniais monitoriaus duomenimis. Monitoriai turi svarbią ypatybę, kurios dėka galima išvengti tarpusavio blokavimo: tik vienas procesas gali būti aktyvus monitoriaus atžvilgiu. Kompiliatorius ypatingu būdu apdoroja monitoriaus procedūrų iškvietimus. Paprastai, kai procesas kviečia monitoriaus procedūrą, tai keletas pirmųjų procedūros instrukcijų patikrina, ar koks nors kitas procesas nėra aktyvus šio monitoriaus atžvilgiu. Jeigu taip, tai kviečiantysis procesas sustabdomas, kol kitas procesas neatlaisvins monitoriaus. Tokiu būdu keleto procesų į monitorių įėjimo išvengimas realizuojamas ne programuotojo, o kompiliatoriaus, kas sumažina klaidų tikimybę.

Paskirstytose sistemose, sudarytose iš kelių procesų, iš kurių kiekvienas turi savo operatyviąją atmintį, semaforai ir monitoriai netinka. Tokiose sistemose sinchronizavimą galima realizuoti tik pranešimų apsikeitimo būdu.

7. Procesų vykdymo planavimas

Procesorius, kaip ir operatyvinė atmintis yra labai brangus resursas, todėl jo efektyvus dalinimas procesams taip pat yra svarbus uždavinys. UNIX yra paskirstyto laiko operacinė sistema, tai reiškia, kad kiekvienam procesui skaičiavimo resursai yra suteikiami tam tikram laiko intervalui, po kurio CPU perduodamas kitam procesui. Maksimalus laiko intervalas, kuriam gali būti suteiktas procesorius, vadinamas laiko kvantu (time quantum arba time slice). Tokiu būdu sukuriama iliuzija, kad keli procesas dirba vienu metu, nors faktiškai vienu metu dirba vienintelis procesas.

Skirtingos programos kelia skirtingus uždavinius operacinei sistemai iš procesų planavimo požiūrio:

- Interaktyvios programos. Šiai klasei priskiriami teksto redaktoriai, komandų interpretatoriai ir kitos programos, tiesiogiai bendraujančios su sistemos vartotoju. Tokios programos daugiausia laiko praleidžia laukdamos vartotojo įvedimo, tarkim, klavišo nuspaudimo arba veiksmo pele. Tačiau, gavus įvedimą, jos turi greitai jį apdirbti, kitaip dirbant su aplikacija nebus komforto. Leidžiamas sistemos reakcijos laikas yra 100 – 200 milisekundžių.
- Foninės aplikacijos. Šiai klasei priklauso programos, kurių darbui nereikia vartotojo įsiterpimo, pvz. kompiliatoriai, skaičiavimo programos ir pan. Šių programų didžiausias reikalavimas greitas skaičiavimas, kad darbas neužtruktų pernelyg ilgai.
- Realaus laiko programos. Šios programos reikalauja papildomų UNIX savybių, kurios garantuotų tam tikros operacijos atlikimo reikiamu laiku. Tarkim video programos reikalauja, kad kadras būtų perpiešiamas tam tikru metu.

Procesų planavimas remiasi t.t. taisyklėmis, remiantis kuriomis procesorius yra perduodamas vienam ar kitam procesui. Anksčiau minėtos programos kelia daug reikalavimų, tačiau šiuolaikinės UNIX sistemos neatsižvelgia į jas visas, o stengiasi rasti aukso vidurį.

Planavimo algoritmai

Pirmas procesas aptarnaujamas pirmiausiai (ang. *first-come, first-served. FCFS).* Tai pats paparasčiausias planavimo algoritmas. Procesoriaus laikas duodamas, laukiančiųjų procesų eilėje, esančiam pirmajam procesui (procesas vykdomas, kol pilnai neužbaigiamas), po to antrajam ir t.t. Taikant šį algoritmą gauname gana didelį vidutinį procesų prastovos laiką.

Pavyzdžiui,

Procesas	**Vykdymo laikas (*ms*)**
P1	24
P2	3
P3	3

Pagal šį algoritmą procesai bus vykdomi šia tvarka: *P1, P2* ir galiausiai *P3*. Pavaizduokime tai, Ganto diagrama.

Apskaičiuokime laukimo laiką. Procesas *P1* lauks 0 ms, *P2* – 24 ms, o *P3* – 27 ms. Taigi vidutinis laukimo laikas yra:

$$t_{vid} = \frac{0+24+27}{3} = 17ms$$

Panagrinėkime atvejį, kai procesai *P2* ir *P3* stovi eilėje prieš *P1* procesą.

Procesas *P1* turės laukti 6 ms, *P2* – 0 ms, *P3* – 3 ms. Šiuo atveju, vidutinis laukimo laikas žymiai sumažėja:

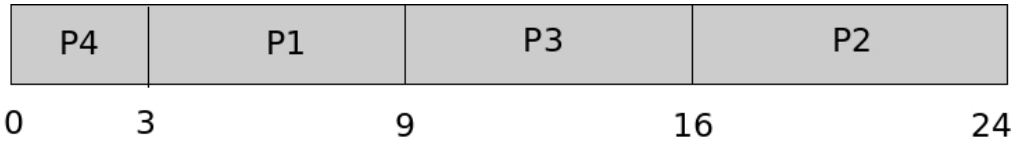
$$t_{vid} = \frac{6+0+3}{3} = 3ms$$

Kaip pastebime, šio algoritmo pagrindinis trūkumas yra, kad jo efektyvumas priklauso nuo procesų eiliškumo. Kuo didesnis procesas yra arčiau eilės priekio, tuo didesnis vidutinis laukimo laikas.

Tobulinant šį algoritmą buvo sugalvotas kitas algoritmas – *trumpiausias darbas atliekamas pirmiausiai (*angl. *shortest-job-first, SJF)*. Pagal šį algoritmą, iš laukiančiųjų eilėje procesų, išrenkamas tas procesas, kuriam įvykdyti reikia mažiausiai laiko. Pavyzdys.

Procesas	**Vykdymo laikas (*ms*)**
P1	6
P2	8
P3	7
P4	3

Remiantis šiuo algoritmu, procesai bus vykdomi šia tvarka: *P4 -> P1-> P3 -> P2*. Nusibraižykime Ganto diagramą.



Apskaičiuosime vidutinį laukimo laiką. Procesas *P1* priverstas laukti 3 ms, *P2* – 16 ms, *P3* – 9 ms, *P4* – 0 ms. Taigi, vidutinis laukimo laikas yra

$$t_{vid} = \frac{3+16+9+0}{4} = 7ms$$

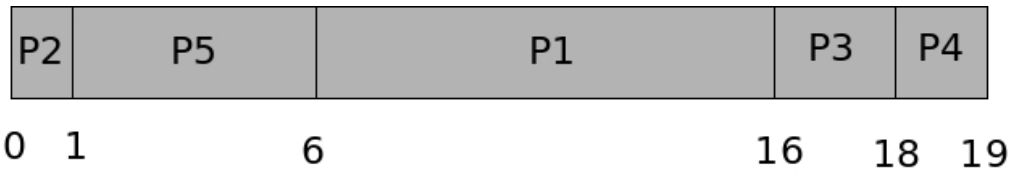
Šio algoritmo privalumas yra tas, kad vidutinis laukimo laikas yra mažiausias, lyginant su kitais algoritmais. Trūkumas – procesorius iš anksto, turi žinoti, kiek laiko procesas bus vykdomas, kad galėtų suplanuoti sekantį veiksmą.

Prioritetais grįstas algoritmas (angl. Priority-based scheduling algorithm). Pagal šį algoritmą kiekvienam procesui priskiriamas prioritetas. Prioritetas – fiksuotas skaičius, kurio reikšmė (priklausomai nuo sistemos) gali būti nuo 0 iki 4095. Bendro sutarimo ar skaičius 0 simbolizuoja aukščiausią prioritetą ar žemiausią nėra. Linux sistemoje 0 simbolizuoja procesą turintį aukščiausią prioritetą. Toliau pateiktame pavyzdyje, sakysime, kad kuo aukštesnis skaičius tuo mažesnis prioritetas.

Pavyzdys.

Procesas	**Vykdymo laikas (*ms*)**	**Prioritetas**
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Kadangi procesas *P2* turi didžiausią prioritetą (mažiausią skaičių prioritetų stulpelyje), tai jis bus vykdomas pirmiausiai, po to procesai *P5 -> P1 -> P3 ->P4*. Nusibraižykime Ganto diagramą.



Apskaičiuokime vidutinį laukimo laiką. Procesas *P1* lauks 6 *ms*, *P2* – 0 *ms*, *P3* – 16 *ms*, *P4* – 18 *ms*, *P5* – 1 *ms*. Taigi vidutinis laukimo laikas

$$t_{vid} = \frac{6+0+16+18+1}{5} = 8{,}2ms$$

Didžiausia problema, naudojant šį algoritmą yra „badavimas“ (angl. starvation). Labai apkrautose sistemose mažus prioritetus, turintiems procesams tenka ilgai laukti, kol juos apdoros procesorius.

Istorijoje buvo toks pavyzdys, kuomet buvo 1973 m. išjungtas IBM 7094 kompiuteris, pastebėjo jog žemiausią prioritetą turintis procesas buvo „nepaliestas“ procesoriaus. Tas procesas į paruoštų procesų eilę buvo patalpintas 1967 m. Šiai problemai išspręsti buvo pasirinkta kas tam tikrą laiko tarpą prie kiekvieno proceso prioriteto pridėti 1 (vienetą). Tokiu būdu, net ir žemiausią prioritetą turintis procesas, taps procesu, turinčiu aukščiausią prioritetą.

Kvantavimo algoritmas (angl. round-robin, RR). Taikant šį algoritmą, procesorius kiekvieną procesą vykdo tam tikrą laiko tarpą – *kvantą*. Laiko kvantas paprastai būna nuo *10* iki *100 ms*. Paruoštų procesų eilė traktuojama, kaip apskrita eilė. Procesorius judėdamas tokia eile kiekvienam procesui skiria vieną laiko kvantą.

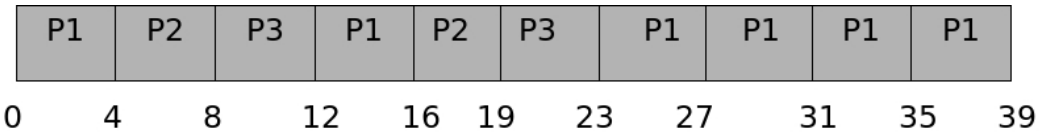
Panagrinėkime pavyzdį.

Tarkime, kad laiko kvantas yra *4 ms*.

Procesas	**Vykdymo laikas (*ms*)**
P1	24
P2	7
P3	8

Kadangi laiko kvantas yra *4 ms*, procesas *P1* gaus pirmąsias 4 ms, po to *4 ms* - *P2* ir *4 ms* – *P3*. Apėjus ratą procesorius vėl imasi proceso P1, P2, P3 tol kol neišsemiama paruoštų procesų eilė.

Nusibraižykime Ganto diagramą, kuri atspindi procesoriaus darbą.



Apskaičiuokime vidutinį laikimo laiką. Pirmas procesas *P1* lauks *0+8+7 =15 ms*, procesas *P2* lauks *4+8 =12 ms*, trečiasis procesas *P3* lauks *8+7=15 ms*.

Taigi, vidutinis laukimo laikas

$$t_{vid} = \frac{15+12+15}{3} = 14ms.$$

8. Atminties valdymas

Pagrindinė atmintis dalinama į dvi dalis: pirma dalis skirta operacinei sistemai, o antra šiuo metu vykdomai vartotojo programai. „Vartotojo“ dalis turi būti paskirstyta tarp daugelio vartotojo programų, t. y. tarp kelių procesų. Ši paskirstymo problema, kurią dinamiškai sprendžia OS vadinama **atminties valdymu**.

Peržiūrint įvairiausias atminties paskirstymo strategijas ir mechanizmus, turi būti patenkinti šie reikalavimai:

Perstūmimas

Multiprograminėje OS pagrindinė atmintis yra padalinama daugeliui procesų. Tam, kad būtų maksimalus procesoriaus apkrovimas, pageidautina turėti gana didelį kiekį pasiruošusių procesų. Tam reikia turėti galimybę pakrauti ir pašalinti aktyvius procesus iš atminties. Reikalavimas, kad pašalinta iš atminties programa pakliūtų lygiai į tą pačią vietą kur buvo anksčiau, yra nepriimtinas, nes sukeltų labai daug apribojimų. Vadinasi pageidautina, kad programa būtų **perstumta** (relocate) į kitą atminties erdvę. Vadinasi iš anksto nėra žinoma į kokią atminties erdvę bus patalpinta programa ir be to programa gali būti perstumta iš vienos erdvės į kitą.

Paprastumo dėlei sakysime, kad procesas užima vieną nepertraukiamą pagrindinės atminties erdvę. Akivaizdu, kad OS būtina žinoti apie procesą valdančią informaciją, steko viršūnę bei įėjimo į programą tašką. Kadangi atminties valdymu užsiima pati OS, tai šiuos atitinkamus adresus ji gauna automatiškai. Tačiau be šios informacijos procesas turi turėti galimybę kreiptis į pačios programos atmintį. Taip vykdymo komandose yra adresai tų komandų, kurios turi būti vykdomos po jų, o nuorodose į duomenis adresai tų duomenų su kuriomis jos dirba. Procesorius ir OS privalo konvertuoti šias nuorodas programos kode į realius fizinius adresus, priklausomai nuo programos vietos atmintyje.

Apsauga

Kiekvienas procesas turi būti apsaugotas nuo nepageidaujamo kito proceso įsikišimo, nepriklausomai nuo to ar tas įsikišimas yra atsitiktinis ar norimas. Vadinasi kitų procesų kodai neturi turėti galimybės į duoto proceso duomenis skaitymui ar rašymui. Tačiau perstūmimo galimybė apsunkina šią apsaugos sąlygą. Programos vieta pagrindinėje atmintyje yra nenuspėjama, o absoliučių adresų tikrinimas kompiliavimo metu yra neįmanomas. Be to

daugelyje programavimo kalbų galimas dinaminis adresų skaičiavimas vykdymo metu. Vadinasi programos vykdymo metu būtina atlikti visų kreipinių į atmintį patikrinimą, tam, kad būti įsitikinusiam, jog šie kreipiniai skirti tik duoto proceso duomenims.

Bendras naudojimas

Kiekvienas apsaugos mechanizmas turi būti pakankamai lankstus tam, kad leistų keliems procesams kreiptis į tą pačią atminties vietą, t.y. naudotis tais pačiais duomenimis.

Loginė organizacija

Pagrindinė atmintis kompiuteryje yra organizuota kaip tiesinė vienamatė erdvė, sudaryta iš baitų sekos.

Fizinė organizacija

Visą kompiuterio atmintį galima paskirstyti į dvi dalis: pirminė (operatyvioji atmintis) ir antrinė (diskinė atmintis arba diskas). Pagrindinė atmintis palaiko labai didelį priėjimo prie duomenų greitį, tačiau gana didele kaina ir nepalaiko ilgalaikio duomenų saugojimo. Antrinė atmintis žymiai lėtesnė, tačiau pigesnė ir duomenys galima saugoti pastoviai. Antrinė atmintis naudojama ilgalaikiam programų bei duomenų saugojimui, o žymiai mažesnės talpos pirminė atmintis naudojama šiuo metu vykdomų programų ir jų duomenų saugojimui.

Tokiu būdu akivaizdu, kad operacinei sistemai skiriama užduotis perduoti informaciją tarp pirminės ir antrinės atminties. Ši užduotis ir sudaro atminties valdymo esmę.

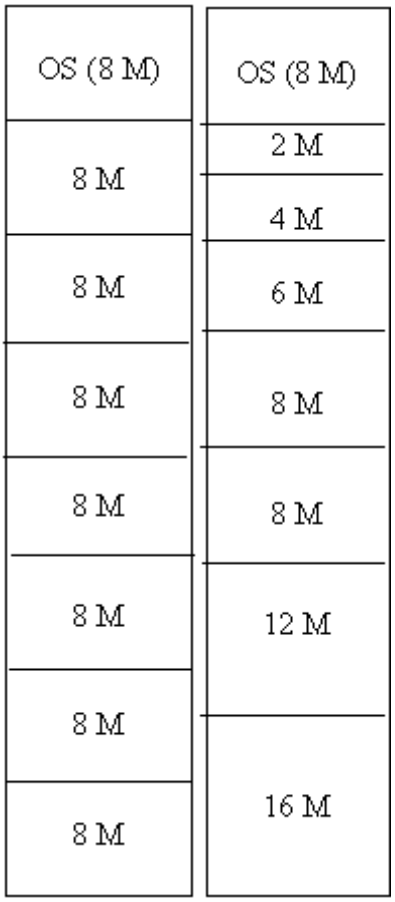
Atminties paskirstymo būdai

Pagrindinė operacija, naudojama atminties valdyme yra efektyvus programos paskirstymas atmintyje, tam, kad ją galėtų vykdyti procesorius.

Galimi keli paskirstymo būdai:

- Fiksuotas paskirstymas;
- Dinaminis paskirstymas.

Fiksuotas atminties paskirstymas



8 pav. Fiksuotas atminties paskirstymas
a) Vienodo dydžio skirsniai b) Skirtingo dydžio skirsniai

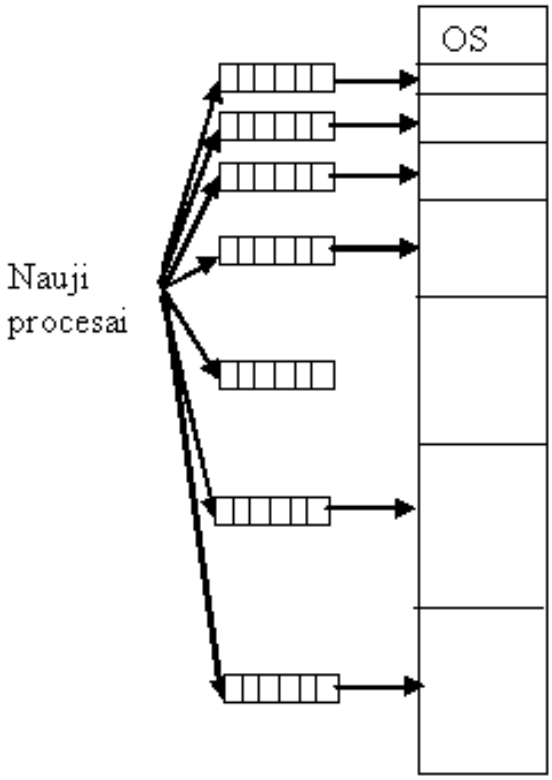
8 pav. pavaizduoti du fiksuoto paskirstymo pavyzdžiai. Kairėje pusėje atmintis yra paskirstyta į vienodo dydžio skirsnius, o dešinėje į skirtingo dydžio skirsnius. Tokiu atveju bet koks procesas neviršijantis skirsnio dydžio gali būti patalpintas į bet kurį laisvą skirsnį. Jeigu visi skirsniai užimti ir nėra nei vieno proceso, kuris yra pasiruošęs ar šiuo metu apdorojamas procesoriaus, OS gali pašalinti bent kurį procesą iš skirsnio ir patalpinti į jį naująjį.

Naudojant vieno dydžio skirsnius galimi šie sunkumai:

- Programa gali būti per didelė vienam skirsniui. Tokiu atveju programuotojas privalo naudoti persidengimus (overlay)
- Atmintis naudojama neefektyviai. Bent kuri programa, nepriklausomai nuo jos dydžio užima visą skirsnį. Taip mūsų pavyzdyje programa, užimanti mažiau nei 1 M, naudos 8 M skirsnį. Ir tokiu atveju liks nepanaudoti 7 M. Tokiu atveju, kai nenaudojama atmintis yra skirsnio viduje vadinama **vidine fragmentacija** (internal fragmentation).

Šiuos trukumus galima pašalinti (tačiau ne pilnai), panaudojant skirtingo dydžio skirsnius. Šiuo atveju, programa užimanti 16 M gali apseiti be perdengimų, o mažo dydžio procesams bus skirti mažesnio dydžio skirsniai.

Patalpinimo algoritmas:



9 pav. Atminties paskyrimas, esant fiksuotam paskirstymui

Tuo atveju, kai skirsniai turi vienodą dydį, paskirstymo užduoties sprendimas yra pakankamai paprastas. Į kokį laisvą skirsnį paklius einamasis procesas yra visiškai nesvarbu. Jeigu visi skirsniai yra užimti procesais, kurie yra nepasiruošę apdorojimui, tai galima bent kurį procesą pašalinti iš atminties, į jo vietą įterpiant naująjį.

Kai skirsniai yra skirtingų dydžių, paprasčiausias būdas yra talpinti kiekvieną procesą į mažiausio dydžio skirsnį, kuriame jis telpa. Tokiu būdu kiekvienam skirsniui yra sudaroma nuosava eilė (9 pav.).

Pagrindinis tokio metodo privalumas yra tas, kad procesai gali būti paskirstyti skirsniams tai, kad vidinė fragmentacija būtų minimali.

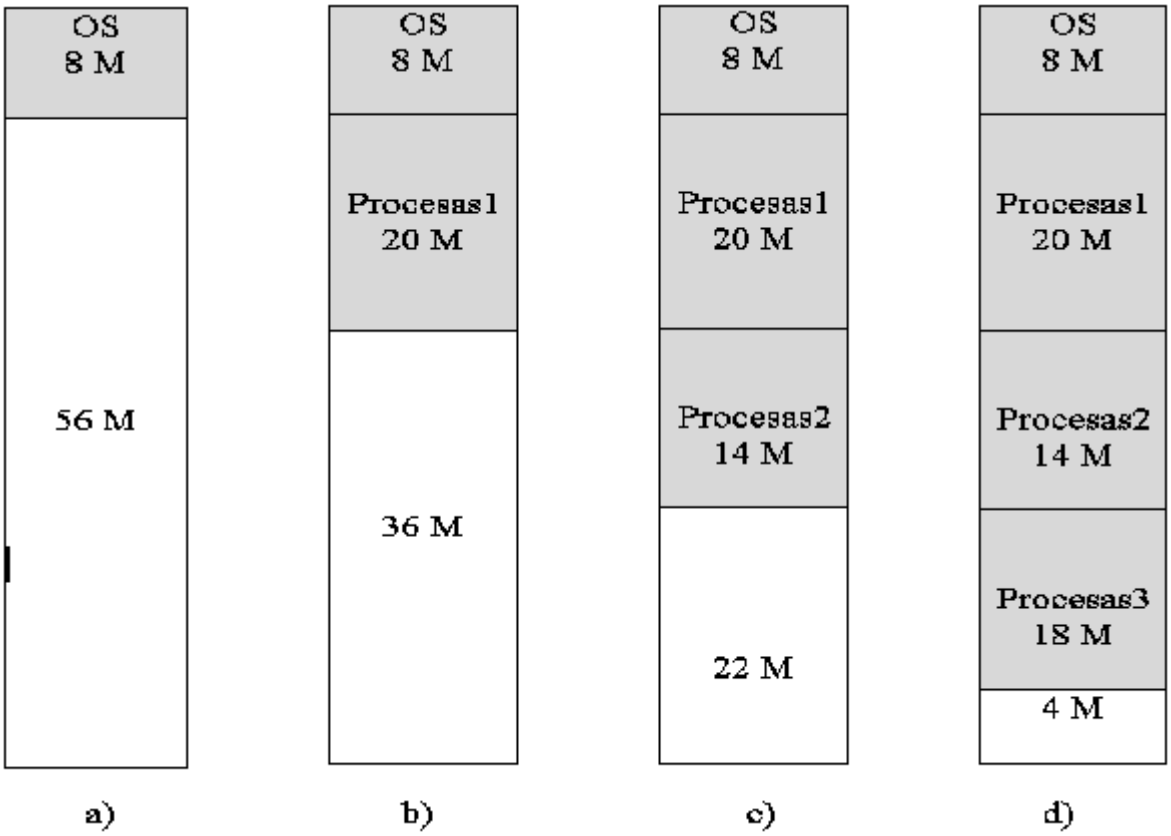
Fiksuoto atminties paskirstymo **trūkumai**:

- Skirsnių kiekis, nustatomas sistemos generavimo metu, apriboja aktyvių procesų kiekį.
- Kadangi skirsnių dydžiai nustatomi iš anksto sistemos generavimo metu, nedideli procesai neefektyviai išnaudoja atmintį. Tuose atvejuose, kai procesų dydžiai iš

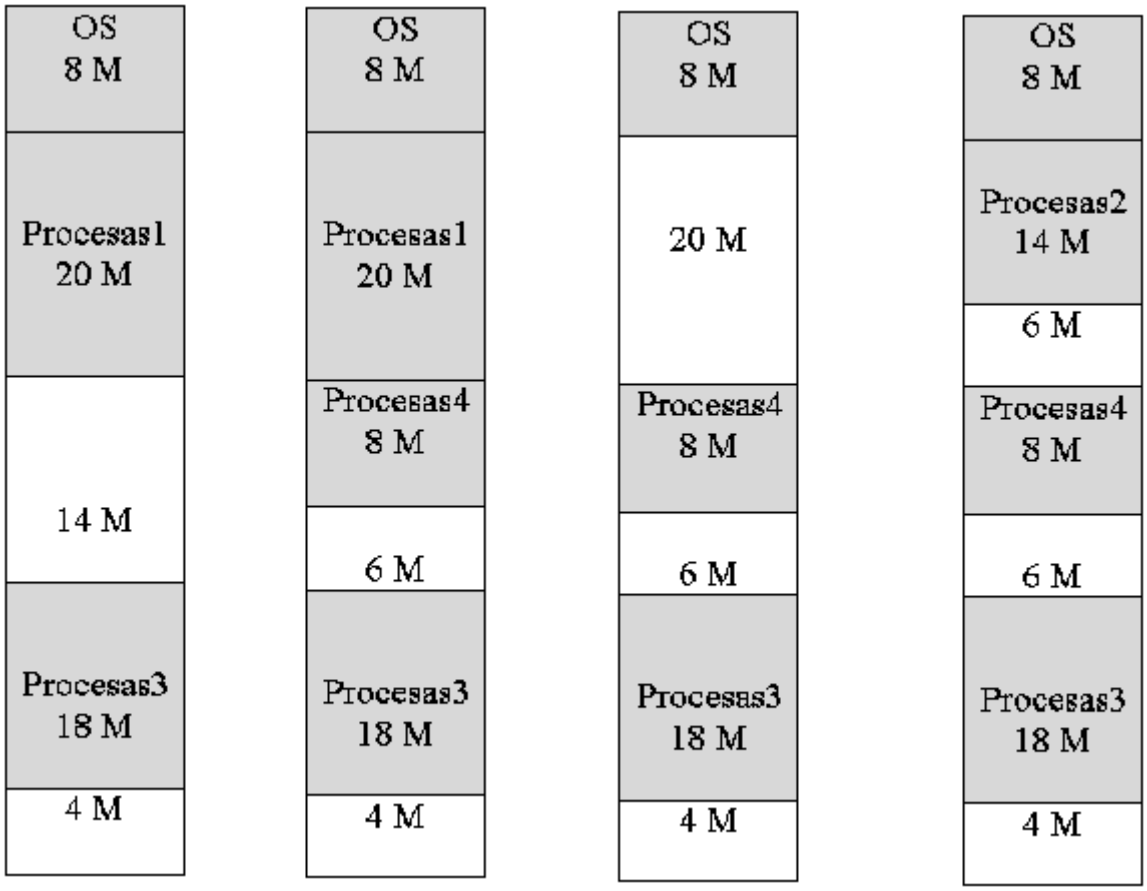
anksto yra žinomi ši schema pasiteisina, tačiau kitose atvejuose efektyvumas pakankamai žemas.

Dinaminis atminties paskirstymas

Tam, kad išvengti sunkumų, susietų su fiksuotu paskirstymu, buvo sukurtas alternatyvus metodas, kuris vadinamas dinaminiu paskirstymu.



10 pav. Dinaminis atminties skirstymas



a) **b)** **c)** **d)**

11 pav.. Dinaminis atminties skirstymas

Esant dinaminiam atminties paskirstymui, atsiranda kintamas skirtingų dydžių skirsnių kiekis. Talpinant procesą į pagrindinę atmintį, jam išskiriama lygiai tiek vietos kiek jis užima. Ir ne daugiau. Kaip pavyzdį išnagrinėkime atvejį pavaizduotą 10 pav.

Pradžioje mūsų naudojama 64 MB atmintis yra tuščia, išskyrus operacinę sistemą (a variantas). Pirmieji trys procesai pakraunami į atmintį pradedant adresu, kuriuo baigiasi OS ir naudoja lygiai tiek atminties kiek jie reikalauja (b)-d) variantai). Po jų lieka 4 MB „skylė“, kuri yra per maža tam, kad patalpinti ketvirtą procesą. Kažkuriuo laiko momentu visi procesai atmintyje tampa neaktyvūs ir operacine sistema pašalina antrąjį procesą (11 pav. a) variantas). Dabar atsiranda vietos ir naujam ketvirtam procesui (b) variantas). Kadangi procesas4 mažesnis už procesą2, tai atsiranda „skylė“. Po tam tikro laiko procesai vėl tampa neaktyvūs, o prosesas2 yra pasiruošęs darbui. Kadangi jam nėra vietos, tai sistema pašalina procesą1, kad atlaisvinti vietą (c) variantas) ir patalpiną procesą2 (d) variantas).

Kaip rodo pavyzdys, šis metodas yra pakankamai geras pradžioje, tačiau pabaigoje susidaro didelis kiekis nedidelių „skylių“ atmintyje. Su laiku atmintis tampa vis labiau fragmentuota ir todėl mažėja jos efektyvumas. Toks reiškinys vadinamas **išorine fragmentacija** (external fragmentation).

Vienas iš metodų, pašalinančių šį reiškinį yra **glaudinimas** (compaction): laikas nuo laiko operacinė sistema perstumia procesus atmintyje taip, kad jei tarpusavyje liestųsi, o laisvos atminties blokas būtų vientisas. Pavyzdžiui po glaudinimo h) variante liktų vienas laisvas 16 M atminties blokas. Tačiau šis metodas reikalauja papildomų didelių laiko sąnaudų.

Atminties adresavimas

Esant fiksuotam paskirstymui visada galima tikėtis, kad procesas paklius į tą patį atminties bloką. Tačiau kitais atvejais ši taisyklė negalioja. Procesas savo darbo metu gali pakliūti į bet kurį laisvą bloką, po kažkiek laiko pašalintas iš jo ir po vėl gali būti pakrautas jau į kitą bloką. Tokia pati situacija matoma ir dinaminiam paskirstyme. Taip (10 pav. Ir 11 pav.) prosesas2 užima skirtingas atminties vietas.

Vykdant glaudinimą procesai taip pat persistumia iš vienos atminties vietos į kitą. Tokiu būdu proceso viduje esantys duomenys ir komandos kiekvieną kartą keičia savo vietą atminties atžvilgiu ir nėra fiksuoti. Sprendžiant šią problemą būtina atskirti adresu tipus.

Loginis adresas tai nuoroda į atminties vietą, nepriklausančią nuo einamosios duomenų vietos atmintyje: prieš tai kaip gauti priėjimą prie atminties ląstelės, būtina transformuoti loginį adresą į fizinį.

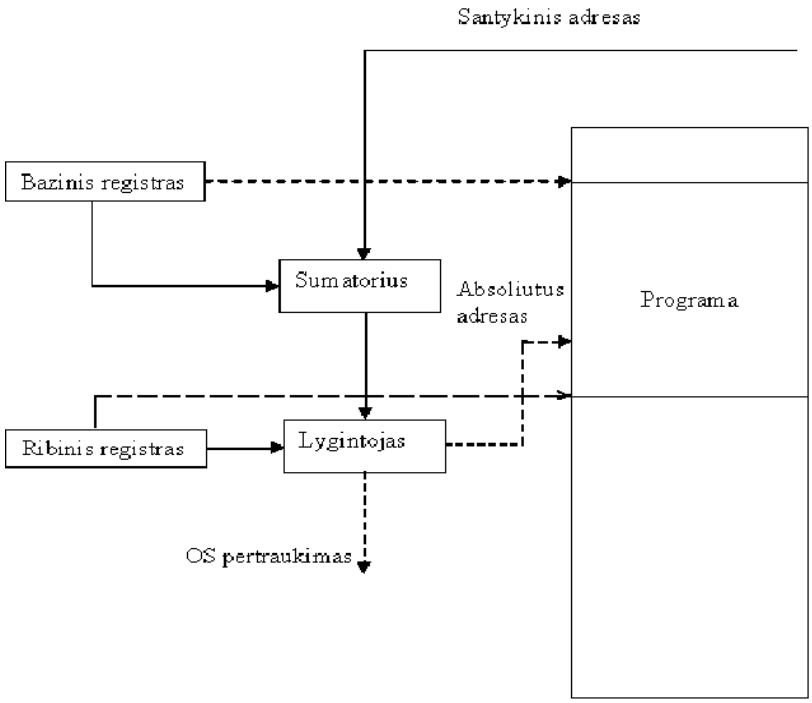
Santykinis adresas, tai atskiras loginio adreso atvejis, kai adresas nusakomas nuo tam tikro iš anksto žinomo taško (paprastai programos pradžios).

Fizinis adresas (arba absoliutusis) tai tiesioginis fizinės atminties ląstelės adresas.

Jei programa naudoja santykinį adresą, tai reiškia, kad visos nuorodos į atmintį einamajame procese vyksta nuo programos pradžios. Vadinasi reikalingi mechanizmai (aparatūros lygmenyje), kurie transformuotų santykinius adresus į fizinius.

12 pav. parodytas dažniausiai naudojamas adresų transliavimo metodas. Kai procesas tampa besivystančiu, į specialųjį procesoriaus registrą (kartais vadinamu baziniu) pakraunamas pradinis proceso atminties adresas. Be to dar naudojamas „ribinis“ (bounds) registras, kuriame saugomas paskutinis programos atminties ląstelės adresas. Skaičiuojant absoliutųjį adresą prie santykinio adreso pridedama bazinio registro reikšmė. Jei gautas adresas priklauso duotam

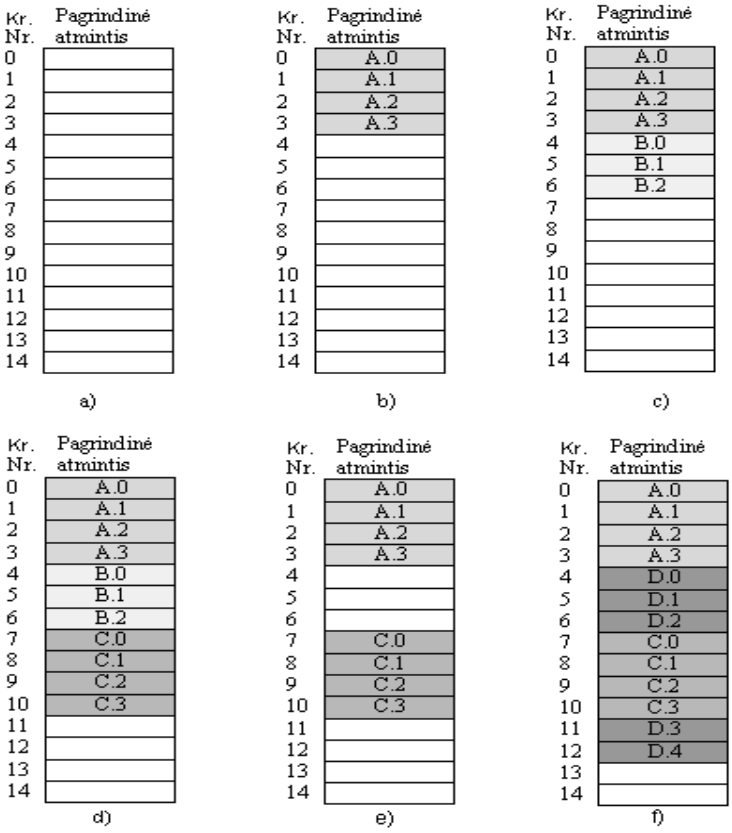
procesui (t.y. ne viršyta „ribinio“ registro reikšmė), tai komanda vykdoma. B Priešingu atveju generuojama atitinkama OS pertraukimo klaida.



12 pav. Absoliutaus adreso paskaičiavimo mechanizmas

Puslapinė atminties organizacija

Tiek vienodo dydžio skirsnių, tiek ir skirtingo dydžio skirsnių atminties organizavimas nepakankamai efektyviai išnaudoja atmintį. Vienu atveju gaunama vidinė fragmentacija, o kitu atveju išorinė. Tarkime, kad visa atmintis yra padalinta į vienodo dydžio blokus, kurie yra sąlyginai nedidelio dydžio. Tada proceso blokai, vadinami **puslapiais**, gali būti susieti su laisvais atminties blokais, kurie bus vadinami **kadrais**. Kiekvienas kadras gali talpinti vieną puslapį. Esant tokiai organizacijai išorinė fragmentacija ne egzistuoja, o vidinė fragmentacija yra minimizuota paskutiniu proceso puslapiu.

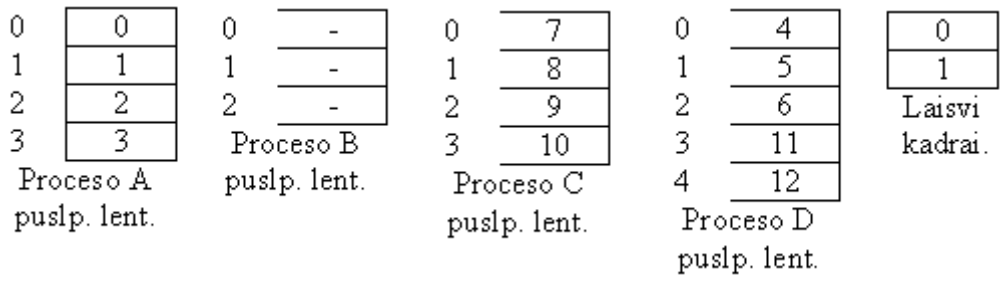


13 pav. Atminties skirstymas puslapiais

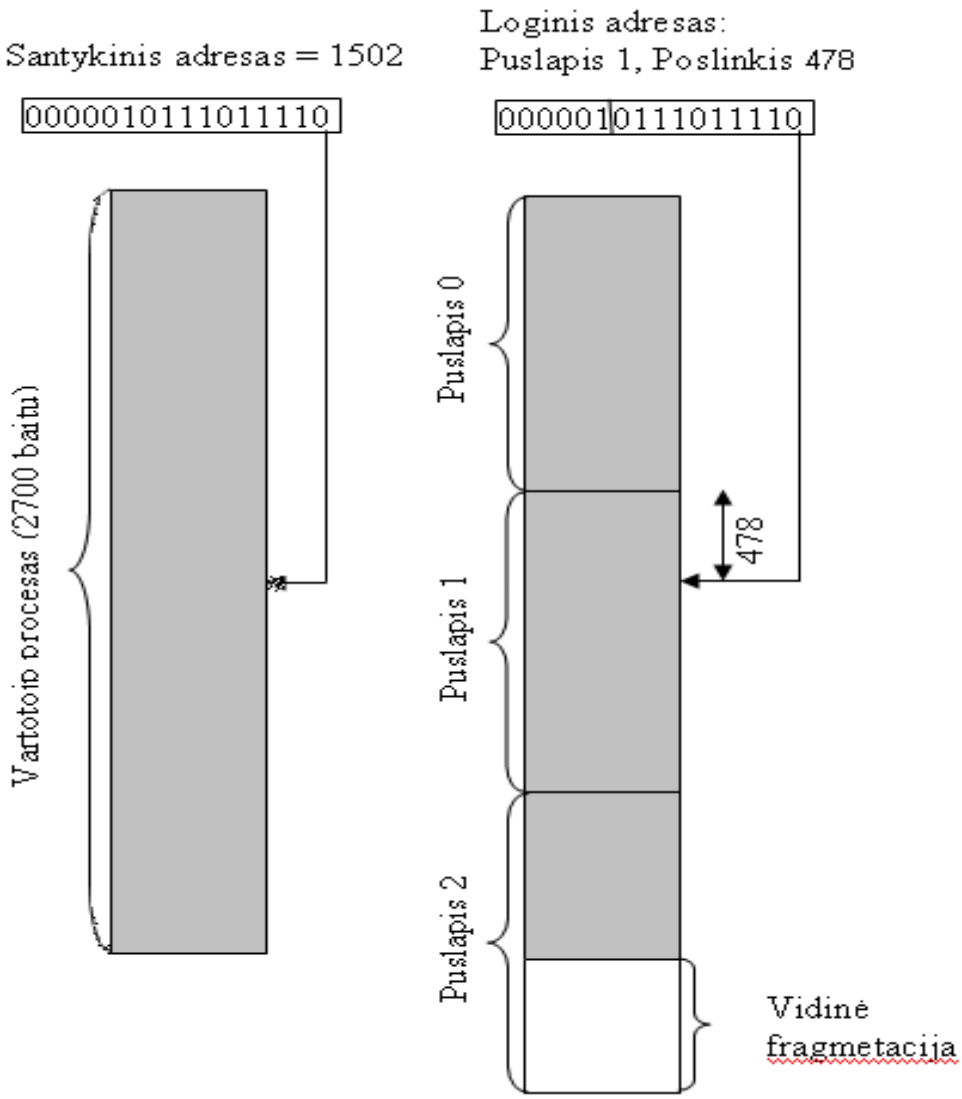
Tam tikru laiko momentu kai kurie kadrai yra naudojami, o kai kurie yra laisvi. OS žino laisvų kadrų kiekį ir numerius. Procesas A, saugomas diske turi 4 puslapius. Kai prasideda jo apdorojimas OS suranda 4 laisvus kadrus ir patalpina į juos procesą A (13 pav. b) variantas). Po to analogiškai pakraunami procesai B (3 puslapiai) ir C (4 puslapiai). Po to procesas B yra pristabdomas ir pašalinamas iš operatyviosios atminties. Dabar reikia pakrauti procesą D, sudarytą iš 5 puslapių. Šiuo atveju mes neturime ištisinės 5 laisvų kadrų sekos. Tačiau, pasinaudojus loginių adresų koncepcija tai padaryti bus galima.

Šiuo atveju vieno bazinio registro neužtenka ir kiekvienam procesui OS palaiko **puslapių lentelę**. Puslapių lentelėje yra nurodyti kardų numeriai kiekvienam puslapiui.

Proceso viduje loginis adresas yra sudarytas iš dviejų dalių: puslapio numerio ir poslinkio jo viduje. 14 pav. parodyti visų procesų puslapių lentelės po to, kai procesas D buvo patalpintas į 4, 5,6, 11 ir 12 kadrą.

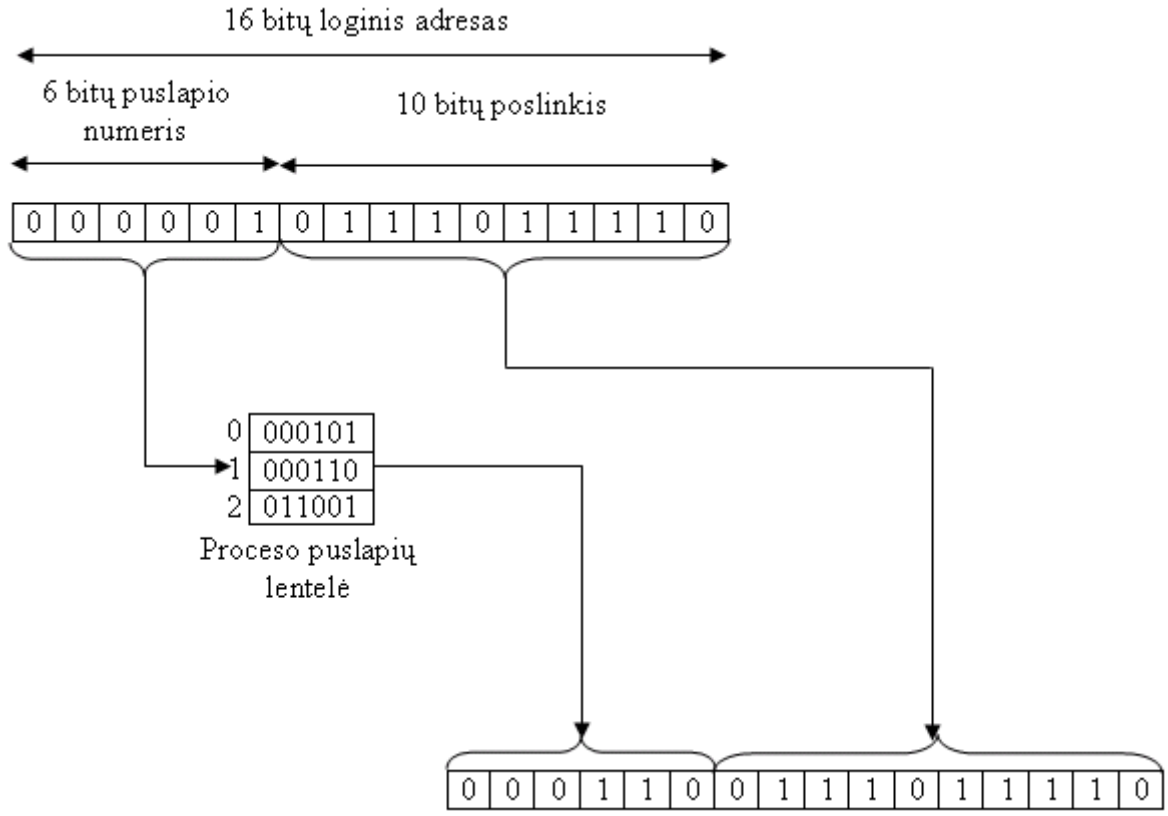


14 pav. Procesų lentelių pavyzdžiai



15 pav. Loginiai adresai

Tam, kad būtų patogiau dirbti su duotu metodu, nustatysime, kad puslapio dydis (o tuo pačiu ir kadro dydis) bus pateiktas 2 laipsniu. Kai puslapio dydis 1 KB, tai poslinkis reikalauja 10 bitų, palikdamas 6 bitus puslapio numeriui. Tokiu būdu mūsų programa gali turėti maksimaliai 2^6=64 puslapius po 1 Kb. Kaip parodyta 15 pav. santykinis adresas 1502 atitinka poslinkį 478 (0111011110) puslapyje 1 (000001), kas duoda binarinį skaičių 0000010111011110 (arba 1502).



16 pav. Puslapinė organizacija

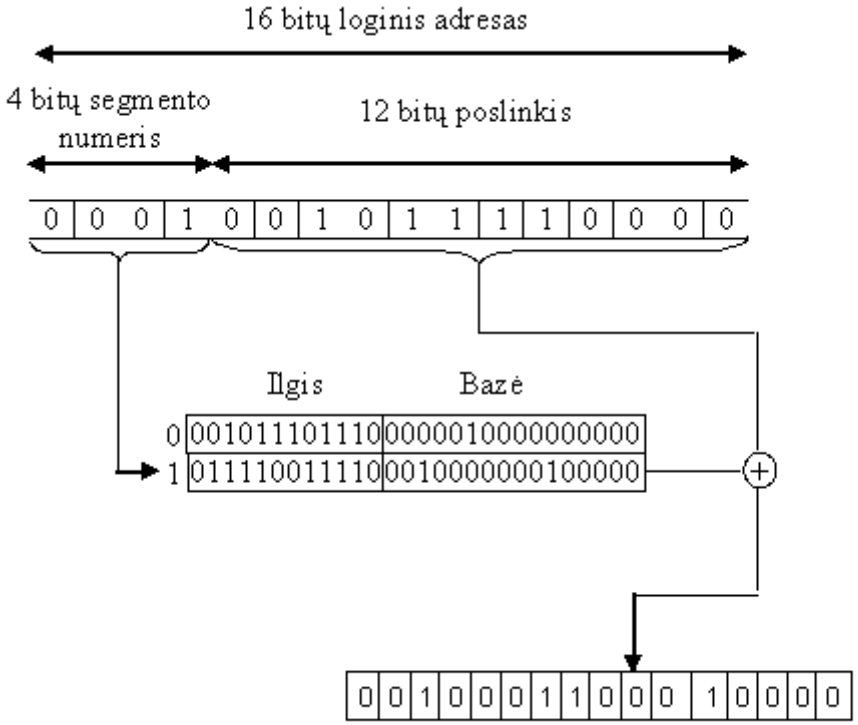
Naudojant puslapius, kurių dydis lygus 2 laipsniu mes gauname gana „skaidrią“ adresaciją, kuri yra patogi tiek OS, tiek ir programuotojui. Tokiu būdu galime labai lengvai konvertuoti adresus darbo metu. Panagrinėkime adresą, sudarytą iš $n+m$ bitų, kuriame kairieji n bitų nurodo puslapio numerį, o dešinieji m bitų – poslinkį. Mūsų pavyzdyje (16 pav.) n=6, o m=10. Tam, kad konvertuoti adresą, būtina atlikti šiuos žingsnius:

- Išskirti puslapio numerį, kuris pateikiamas n kairiaisiais bitais
- Naudojant puslapio numerį kaip indeksą puslapių lentelėje, surasti kadro numerį k.
- Pradinis fizinis kadro adresas yra kx2^m ir dominantis fizinis adresas bus šis skaičius plius poslinkis.

Mūsų pavyzdyje turime loginį adresą 0000010111011110, nusakomas puslapio numeriu 1 ir poslinkiu 478. Tarkime, kad šis puslapis yra pagrindinės atminties 6-ame (dvejat. sk. 000110) kadre. Tokiu būdu mūsų fizinis adresas yra 6 kadras ir 478 poslinkis, t. y. 0001100111011110.

Segmentacija

Alternatyvus atminties skirstymo metodas yra segmentacija. Šiuo atveju programa ir susieti su ja duomenys skirstomi į aibę **segmentų**. Nors ir egzistuoja segmento dydžio riba, tačiau visi segmentai nebūtinai gali būti vienodo dydžio (skirtingai nei puslapiai). Kaip ir puslapiavimo atveju, loginis adresas sudarytas iš dviejų dalių: segmento numerio ir poslinkio.



17 pav. Segmentacija

Naudojant skirtingo dydžio segmentus, šis metodas yra panašus į dinaminį atminties paskirstymą, tačiau čia segmentai gali užimti kelis atminties skirsnius. Jei puslapiai yra „nematomi“ programuotojo, tai segmentai, kaip taisykle, yra „matomi“ ir naudojami talpinant kodą ir duomenis į skirtingus segmentus. Programuotojui reikia stengtis tik neviršyti segmento dydžio.

Konvertuojant loginį adresą į absoliutųjį segmentacijoje yra nedideli skirtumai, lyginant su puslapiniu organizavimu. Kaip ir puslapinėje organizacijoje, kiekvienas procesas turi segmentų lentelę ir laisvų blokų sąrašą. Kiekvienas lentelės įrašas savyje turi pradinį segmento adresą ir jo dydį. Išnagrinėkime adresą $n+m$ pavidalo. Mūsų atveju 17 pav. pavaizduota, kad n=4, o m=12. Tokiu būdu maksimalus segmento dydis bus lygus 2^{12}=4096. Adreso transliavimui būtina atlikti šiuos žingsnius:

- Išskirti segmento numerį, kuris pateikiamas n kairiaisiais bitais.

- Naudojant puslapio numerį kaip indeksą segmentų lentelėje, surasti pradinį fizinį šio segmento adresą.
- Sulyginti poslinkį, pavaizduotą kaip m dešiniųjų bitų su segmento dydžiu. Jei poslinkis didesnis už dydį, tai adresas ne korektiškas.
- Ieškomas fizinis adresas, tai suma segmento pradžios fizinio adreso ir poslinkio.

Mūsų pavyzdyje loginis adresas 0001001011110000 pateikiamas, kaip 1 segmento numeris ir poslinkis 752. Tarkime, kad šis segmentas talpinamas pagrindinėje atmintyje pradedant nuo adreso 0010000000100000. Tada dominantis mus fizinis adresas bus lygus 0010000000100000+001011110000=0010001100010000 (17 pav.).

Virtuali atmintis

Išnagrinėję puslapinę ir segmentinę atminties organizaciją, galime pateikti šias išvadas:

- Visi kreipiniai į atmintį proceso ribose pateikiami loginiu adresu, kuris dinamiškai transliuojamas į fizinį adresą vykdymo metu. Tai reiškia, kad procesas gali būti pašalintas iš atminties į diską ir savo vykdymo metu gali būti skirtingose atminties vietose.
- Procesas yra padalintas į dalis (puslapius, segmentus), kurie ne būtinai gali būti talpinami pagrindinėje atmintyje vienu nepertraukiamu bloku. Toks mechanizmas yra palaikomas dėl dinaminio adresų transliavimo, naudojant puslapių ir segmentų lenteles.

Tokiu būdu, *mes galime teigti, kad visų proceso puslapių ar segmentų būvimas pagrindinėje atmintyje vienu metu* ***nėra būtina***.

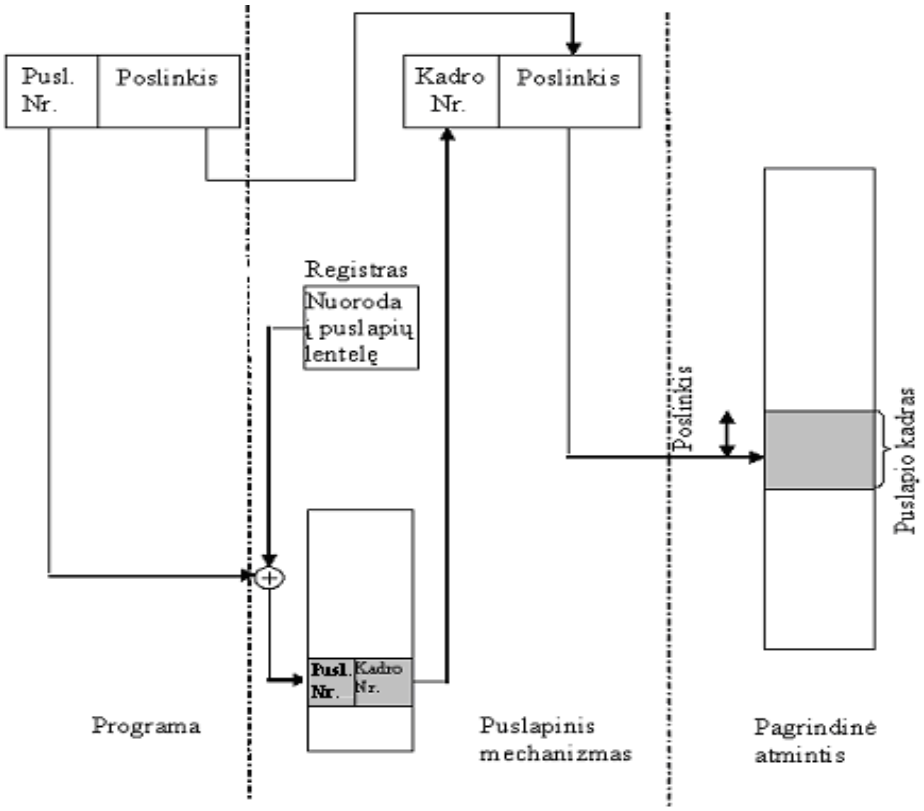
Panagrinėkime, kaip tai gali įvykti.

Paprastumo dėlei puslapį ar segmentą vadinsime **bloku**. Tarkime, kad procesas išrenkamas apdorojimui ir talpinamas į pagrindinę atmintį. OS pradeda talpinti proceso kelis blokus, pradedant nuo bloko, kuriame yra programos pradžia. Dalis proceso patalpinama į pagrindinę atmintį ir vadinama rezidentine proceso dalimi. Proceso apdorojimo metu viskas vyksta taip, kad visi kreipiniai vyksta tik į rezidentinę proceso dalį. Naudojant blokų lenteles, visada galima nustatyti ar blokas į kurį vyksta kreipimas yra pagrindinėje atmintyje ar ne. Jeigu procesorius susiduria su loginiu adresu, kurio nėra pagrindinėje atmintyje, procesas užblokuojamas ir jo darbas tęsiamas tik tada, kai į atmintį pakraunamas reikalingas blokas, turintis tą loginį adresą.

Tokia technologija leidžia mums daryti šias išvadas:

- **Pagrindinėje atmintyje galime saugoti didesnį kiekį procesų**. Kadangi į atmintį talpinama tik tam tikra proceso blokų dalis, tai leidžia mums efektyviau išnaudoti atmintį, didinant aktyvių procesų skaičių pagrindinėje atmintyje.
- **Procesas gali būti didesnis už visą pagrindinę atmintį.** Panaikintas labai didelis programavimo apribojimas. Dažniausiai programuotojas turi apskaičiuoti kiek pagrindinės atminties užims jo programa ir imtis tam tikrų priemonių, skaidant programą į dalis, kurios tilps į atmintį. Naudojant virtualią atmintį, ši funkcija perduodama OS ir aparatūrai. Programai (programuotojui) dabar prieinamas didelis atminties kiekis, kuris apribojamas disko dydžiu.

Bazinis adresų transliavimo mechanizmas tai transliavimas virtualaus adreso į realų. 18 pav. pademonstruotas šis mechanizmas.



18 pav. Virtualaus adreso transliavimas į realų

Vykdant kokį nors procesą startinis jo puslapio lentelės adresas saugomas registre, o puslapio numeris iš virtualaus adreso naudojamas kaip elemento indeksas, kuriame ieškomas atitinkamas kadro numeris. Po to šis numeris apjungiamas su poslinkiu iš virtualaus adreso tam, kad gauti mus dominantį realų atminties ląstelės adresą.

10. Virtualioji atmintis

Jei procesorius vienu metu gali vykdyti tik vieną instrukciją (komandą), tai kam reikia laikyti visas programos komandas atmintyje kol jos dar nėra vykdomos? Prieš tai peržvelgėme persidengiančias struktūras, kuriose tik dalis programos bet kuriuo momentu yra laikoma atmintyje. Ši persidengimo idėja vėliau ir peraugo į virtualios atminties ideologiją.

Pats žodis virtualus reiškia “ne faktinis”. Programuotojui arba vartotojui virtuali atmintis reiškia tą patį kaip reali atmintis, bet faktiškai tai nėra reali atmintis.

Virtualios atminties panaudojimas

Virtuali atmintis yra daloma į dvi komponentes. Pirmoji yra tiksliai sutapatinama su realia atmintimi, jos kiekiu sistemoje, ir fiziškai yra saugoma realioje atmintyje. Joje yra sritis rezidentinės operacinės sistemos saugojimui bei kintama sritis , skirta laikymui kintančių programų sričių. (tai puslapių erdvė – puslapių pulas).

Antra virtualios atminties komponentė apima atminties sritį, kuri užeina už realios atminties ribų. Fiziškai ji yra saugoma išoriniame, puslapiais sudalintame įrenginyje ir apima taikomąsias programas. Operacinė sistema yra patalpinta realioje atmintyje. Taikomosios programos yra išoriniame įrenginyje. Išrinkti taikomųjų programų puslapiai yra kilojami tarp išorinio įrenginio ir realios atminties puslapių pulo.

Tradiciškai , operacinės sistemos atminties valdymo paprogramė yra susijusi su realios atminties išskyrimu. Esant virtualios atminties schemai, ekvivalentiškas modulis išskiria atmintį išoriniame įrenginyje. Ši atmintis gali būti suskirstyta į fiksuoto ilgio dalis, kintamo ilgio dalis, segmentus, puslapius arba bet kokiais kitais vienetais. Keitimasis puslapiais tarp atminties bei išorinio įrenginio yra grynai operacinės sistemos funkcija.

Virtuali atmintis yra modelis, kuris apima sritį, skirtą operacinei sistemai ir taikomosioms programoms skirtas sritis. Operacinė sistema bei išrinkti puslapiai užima realią atmintį. Taikomosios programos yra saugomos išoriniame įrenginyje.

Virtualios atminties adresacija

Komandos, kurios vykdomos virtualioje atmintyje yra identiškos komandoms, vykdomoms įprastoje sistemoje. Operandai turi reliatyvius adresus (bazė plius poslinkis).

Taigi, po to kai komanda yra įkeliama vykdymui, kontrolės įrenginys išplečia adresą, prie poslinkio pridėdamas bazinio adreso turinį. Įprastose sistemose baziniame registre saugomas programos užkrovimo į realią atmintį pradžios adresas. Virtualios atminties atveju, baziniame adrese laikomas programos užkrovimo į virtualią atmintį adresas. Dinaminis adreso transliavimas, pradedamas kai programa yra įkeliama į virtualią atmintį. Operacinė sistema išskiria sritį išoriniame įrenginyje ir atžymi virtualius segmentų bei puslapių adresus programos segmentų bei puslapių lentelėse. Vėliau, įkėlus puslapį į realią atmintį, puslapio realus adresas atžymimas puslapių lentelėje.

Reikia pažymėti, kad puslapis turi rastis realioje atmintyje tam, kad procesorius galėtų atlikti puslapyje esančias komandas. Komandos vykdymo metu kontrolės įrenginys (bendru atveju) prideda bazinio registro turinį prie poslinkio tam, kad gauti adresą virtualioje atmintyje. Verčiant virtualų adresą į realų, techninė įranga:

1. Pasinaudoja segmentų lentele – tam reikalui naudojami vyriausi adreso bitai.
2. Atranda programos puslapių lentelę naudojant nuorodą iš segmentų lentelės.
3. Pasiekia puslapių lentelę ir atranda realų bazinį adresą – tam naudojami adrese esantys viduriniai bitai.
4. Prie puslapio bazinio realaus adreso prideda poslinkį, kurį nusako jauniausi adreso bitai.

Daugumoje sistemų procesas yra greitinamas naudojant specialios paskirties registrus.

Puslapio trūkumas

Kai virtualus adresas rodo į puslapį, kurio nėra realioje atmintyje, kyla situacija, suprantama kaip puslapio trūkumo (page fault) ir pradedama puslapio įkėlimo operacija. Jei naujam puslapiui patalpinti nėra laisvos atminties, kai kurie kiti puslapiai turi būti iš realios atminties iškelti. Paprastai iškėlimui yra renkamas puslapis, į kurį paskutiniu metu buvo mažiausiai kreiptasi.

Puslapių įkėlimas tuo metu, kai jų prireikia yra vadinamas puslapiavimu kilus poreikiui (demand paging). Išankstinis puslapiavimas (prepaging) apima nustatymą, kokio puslapio reiks ir jo įkėlimą į atmintį prieš tai, kol jo realiai prireiks. Daugumoje išankstinio puslapiavimo algoritmų yra laikoma, kad segmentai apima logiškai surištą kodą, taigi, jei šiuo momentu yra

vykdomos 1 lape esančios instrukcijos (komandos), tai skaitoma, kad bus didelė tikimybė, kad toliau reikės 2 lapo. Nors tai nėra tobula, bet tai leidžia pagreitinti programos vykdymą.

Perkrovimai

Kai reali atmintis tampa pilna, užklausa naujam puslapiui reiškia, kad kitas puslapis turi būti iškeltas. Jei tai vyksta dažnai, tai sistema visą savo laiką gali ir paskirti tik puslapių kilojimui iš atminties arba į atmintį, ir labai mažai beliks laiko naudingam darbui.

Ši problema vadinama perkrovimu (trashing). Tai gali stipriai sumažinti sistemos efektyvumą. Paprasčiausias sprendimas tokiu atveju yra programos ar dviejų programų iškėlimas iš realios atminties, kol sistema vėl ims dirbti normaliai. Realesnis sprendimas būtų realios atminties kiekio padidinimas.

Prielaidos ir savybės

Procesas gali būti pakrautas į atmintį dalimis (puslapiais arba segmentais).

Atminties adresai gali būti apskaičiuojami (transliuojami) dinamiškai.

Didesnis skaičius procesų gali būti vykdomas vienu metu.

Kiekvienas procesas gali turėti daugiau (virtualios) atminties nei realiai yra.

Virtualios atminties realizacijos mechanizmai

Puslapiais organizuojama virtualioji atmintis.

Vykdymo pradžioje pakrovėjas talpina į atmintį vieną puslapį. Toliau generuojami puslapių pareikalavimo pertraukimai. Formuojamas proceso rezidentinis rinkinys. Kai realioje atmintyje nebėra nevieno laisvo puslapio, OS vykdo puslapių apsikeitimą tarp pagrindinės ir diskinės atminties.

Kiekvieno proceso adresų transliavimas vyksta naudojant atskiras puslapių lenteles. Specialus puslapių lentelės registras saugo nuorodą į aktyvaus proceso lentelės adresą atmintyje. Keičiantis procesui keičiasi jo kontekstas (t.y. registrų turinys). Adresų transliavimą vykdo elektroninės (procesoriaus) schemos.

Puslapių apsikeitimo strategijos

LRU - puslapis, mažiausiai naudotas pastaruoju metu (least recently used).

NRU - nenaudotas pastaruoju metu.

FIFO - eilės metodas: keičiamas puslapis, kuris ilgiausiai buvo atmintyje.

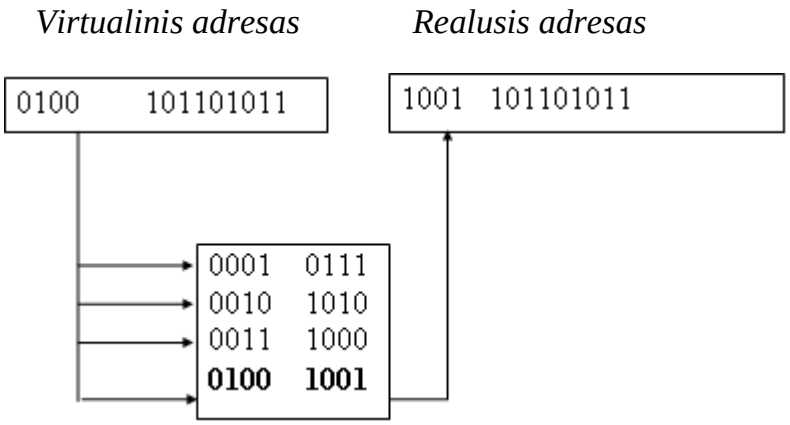
Clock arba *second chance* - eilė kaip žiedinis sąrašas su panaudojimo bitu.

NWPF - išankstinis laisvų atminties blokų paruošimas puslapių pakrovimui.

Elektroniniai virtualios atminties realizacijos mechanizmai

TLB - translation lockaside buffer- atskiras transliavimo buferis.

Associative mapping - asociatyvus atvaizdavimas.



19 pav. Virtualaus adreso transliavimas į fizinį.

Segmentais skirstoma virtualioji atmintis

Palengvina dinaminės atminties panaudojimą.

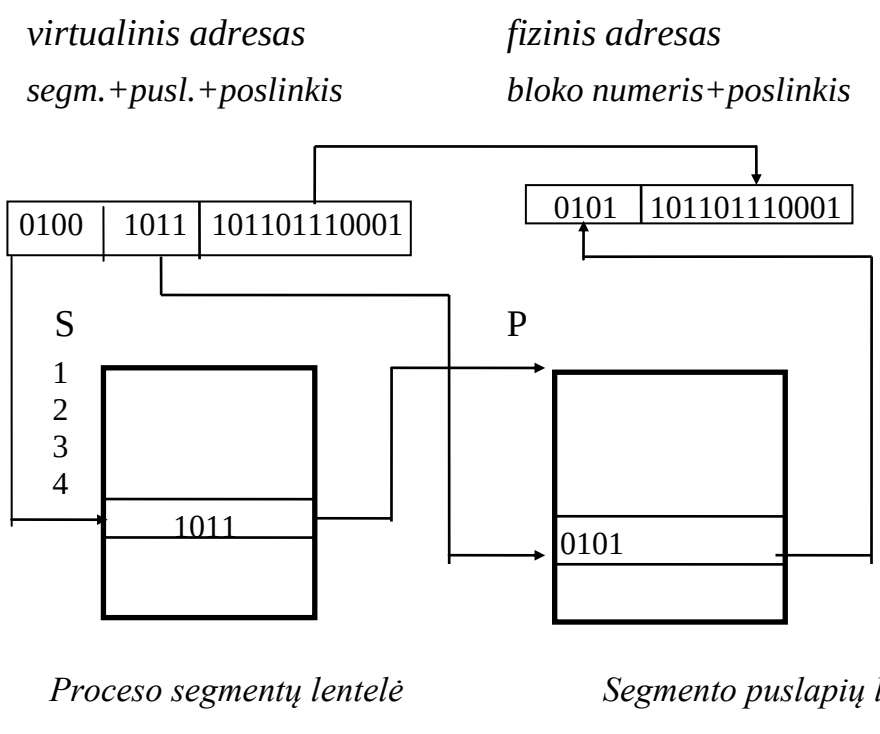
Leidžia bendrai naudotis kodu ir duomenimis.

Geresnė adresų lokalizacija, nes segmentais atvaizduota proceso struktūra.

Segmentų lentelės eilutė - *segmento deskriptorius.* Jo tipinis turinys:

bazinis segmento adresas; segmento max dydis; segmento požymių bitai, t.y. būvimo realioje atmintyje (y,n), panaudojimo (u), apsaugos (r,w,e).

Kombinuotas puslapių ir segmentų metodas (MS-Windows, OS/2)



20 pav. Virtualaus adreso transliavimas į fizinį, puslapių – segmentų struktūroje

11. Pagrindiniai Unix failų tipai

Kas yra failas?

Failo supratimas Unix OS yra žymiai platesnis: failas yra bet kuris šaltinis iš kurio duomenys gali būti skaitomi, arba bet kuris "gavėjas", galintis priimti duomenis. Tokiu būdu, terminas *failas* nurodo ne tik duomenų saugyklą diskinio failo pavidale, bet kartu reiškia ir bet kurį fizinį įtaisą. Failas yra ir klaviatūra (įvedimo šaltinis), ir displėjus (atvaizdavimo įtaisas), ir kiekvienas spausdintuvas.

Unix'e yra trys pagrindiniai failų tipai: paprastieji failai, katalogai ir specialūs failai.

PAPRASTIEJI failai yra tai, ką mes suprantame failu siaurąja prasme. Paprastąjį failą sudaro diske (rečiau juostoje) saugomi duomenys. Tai failai, su kuriais dažniausiai tenka dirbti. Pvz., kai jūs teksto redaktoriumi tvarkote dokumentą, tai jūs naudojate du paprastuosius failus: dokumento ir redaktoriaus programos.

Kitas failo tipas – KATALOGAS. Katalogas tai failas, kuriame saugoma kitų failų pasiekimui reikalinga informacija. Tik konceptualiai į katalogą įeina kiti failai. Vartotojas nekuria katalogų ir netalpina juose failų. Jis tik naudoja komandas, kurios paveda tuos darbus Unix'ui. Jeigu jūs pažiūrėsite į katalogo 'vidų', tai pamatysite 'miglotą' informaciją. Geriausias būdas suprasti katalogo sąvoką - įsivaizduoti, kad tai yra rodyklių į kitus failus visuma. Katalogas gali turėti rodykles į kitus katalogus. Tai leidžia organizuoti failus į hierarchinę sistemą. Kaip matysime vėliau, Unix failų sistema yra organizuota kaip didžiulė failų ir katalogų hierarchija.

Paskutinis failų tipas - tai SPECIALŪS, arba PRIETAISŲ failai, kurie yra vidinė fizinių prietaisų reprezentacija. Pvz., klaviatūra, displėjus, spausdintuvas, disko tvarkyklė - visi jūsų sistemos prietaisai gali būti pasiekiami kaip failai. Pasiusti duomenis į ekraną, ar į kitą terminalą, galima paprasčiausiai juos rašant į failą, kuris reprezentuoja atitinkamą prietaisą.

Pvz., *ls > /dev/pts/u*

Unix OS yra tokie failų tipai:

Tipas	Žymėjimas
Regular Directory	(-) (d)
Hard Links	(-)
Soft Links	(l)
Block special	(b)
Character special	(c)
Socket	(s)
Named pipe	(p)

2 lentelė. UNIX OS failų tipai ir žymėjimas

Kiekvienas sistemos failas turi taip vadinamą inode bloką, kuriame yra informacija apie failą. Inode blokas turi tokią struktūrą:

```
struct stat {
dev_t   st_dev; ino_t   st_ino;     /*      device inode resides on */
u_short st_mode; short              /*      this inode's number */
st_nlink; short   st_uid; short     /*      protection */
st_gid; dev_t   st_rdev;            /*      number or hard links to the file */
device */                           /*      user-id of owner */
off_t   st_size;                    /*      group-id of owner */
time_t  st_atime;                   /*the device type, for inode that is
int     st_sparel;
time_t  st_mtime;                   /* total size of file */
int     st_spare2;                  /* file last access time */
time_t  st_ctime;
int     st_spare3;                  /* file last modify time */
long    st_blksize;
long    st_blocks;                  /* file last status change time */
long    st_spare4;
u_long  st_gennum;                  /* optimal blocksize for file system i/o ops */
};                                  /* actual  number of blocks allocated */

                                    /* file generation number */
```

Pagrindiniai šios struktūtos laukai st_mode (the permission bits), st_uid - UID, st_gid - GID, ir st_*time (assorted time fields).

Failo laikai

Unix įrašo į *inode* tris failo laikus, kurie vadinami ctime, mtime, ir atime.
ctime rodo paskutinį inode bloko keitimo laiką; mtime rodo paskutinį failo modifikavimo laiką; atime rodo paskutinį kreipimosi į failą laiką.
Inod'o ctime kinta visuomet, kai į failą yra rašoma, arba kinta jo apsaugos ar nuosavybės bitai. Paprastai ctime geriau pažymi failo modifikavimą, negu laukas mtime.
Failo laikai yra svarbūs ir įvairiais būdais naudojami sistemos administravimo metu. Pvz., darant diskų kopijas.

Tekstiniai ir dvejetainiai failai, bitai ir baitai

Paprasti failai gali būti labai įvairūs, kadangi labai įvairūs gali būti juose saugomi duomenys. Tačiau, plačiąja prasme, paprastus failus galima skirti į dvi grupes: tekstinius failus ir dvejetainius failus.

TEKSTINIAI failai yra paprasti failai, kuriuose saugomi tik ASCII simboliai: duomenys, kuriuos galima surinkti klaviatūros pagalba. Trumpai tariant, skirtingus ASCII simbolius reprezentuoja 128 skirtingi kodai: didžiosios ir mažosios raidės, skaičiai, tarpo simbolis, tabuliacijos simbolis, skyrikliai, o taip pat valdymo simboliai. Praktinė taisyklė: jei paprastame faile saugomi duomenys, kuriuos jūs galite suprasti ir tvarkyti teksto redaktoriaus pagalba, tai tas failas yra tekstinis. Priešingu atveju yra dvejetainis failas.
Tekstinio failo kiekvienas ASCII kodas saugomas viename baite. Tokiu būdu failas, turintis 1000 simbolių, yra 1000-ties baitų ilgio.

Griežtai kalbant, ASCII kodas naudoja septynis bitus vienam simboliui (baite yra 8 bitai). Kiekvieno baito aštuntas bitas (kairysis) visuomet turi nulinę reikšmę.

Apibendrinant galima pasakyti, kad tekstinis failas saugo ASCII simbolius ir naudoja kiekvienam simboliui septynis baito bitus. Dvejetainis failas saugo duomenis, kurių kodams naudojami visi aštuoni kiekvieno baito bitai.

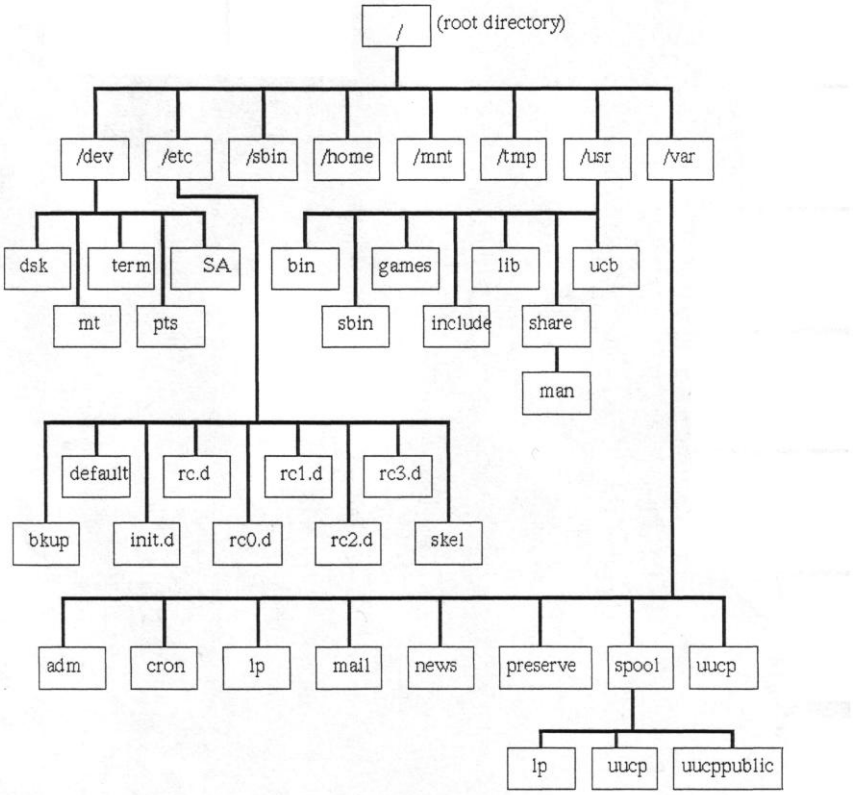
Katalogai ir Pakatalogiai

Failų organizavimui į hierarchinę sistemą naudojami katalogai. Mes grupuojame failus į grupes ir saugome kiekvieną grupę atskirame kataloge. Kadangi katalogas taip pat yra failas, tai kataloge gali būti saugomi kiti katalogai.

Medžio pavidalo failų sistema

Unix sistemą sudaro daugybė failų, kurie organizuojami į tam tikrą struktūrą, naudojant katalogus ir pakatalogius. Unix failų sistema remiasi viena pagrindiniu katalogu, vadinamu ROOT DIRECTORY (šakninis katalogas). Šakninis katalogas yra protėvis visų kitų sistemos katalogų.

Pateiktoje schemoje parodytas Unix failų sistemos turinys. Tokius katalogus naudoja Berkeley Unix pagrindu sukurtos sistemos. System V, arba kitu pagrindu sudarytos sistemos gali nežymiai skirtis nuo čia pateiktos. Tai ką jūs matote yra tik nuogas skeletas. Visos Unix sistemos turės ir kitus, sistemų menedžerių sukurtus katalogus.



21 pav. Direktorijų medis UNIX OS

Nesunku pastebėti, kad Unix failų sistema panaši į apverstą medį. Žodis *root* buvo parinktas tam, kad pabrėžti pagrindinę medžio liemens dalį. Kadangi šakninio katalogo vaidmuo yra labai svarbus, tai šis vardas dažnai turi būti naudojamas komandose. Todėl buvo įvestas žodžio *root* sutrumpinimas / (slash).

Pvz.,

ls komanda parodys visus specifikuoto katalogo failų vardus.

Visus šakninio katalogo failus parodo komanda

ls /

Šaknyje esantis failas, arba katalogas nurodomi rašant / prieš atitinkamą vardą. Pav., formalus etc katalogo vardas yra /etc. Katalogas, arba failas, esantis kitame kataloge, nurodomi atskiriant vardus simboliu /. Pav., kataloge /etc yra passwd failas. Formalus šio failo vardas yra /etc/passwd. Panašiai, jeigu /usr kataloge yra kitas katalogas dict, o jame yra failas words, tai formalus šio failo vardas yra /usr/dict/words.

Simbolis / turi dvi skirtingas reikšmes. Failo vardo pradžioje jis reiškia šakninį katalogą. Failo vardo viduryje jis atlieka skyriklio vaidmenį.

Gerai supratus pateiktą Unix katalogų medį, galima pastebėti, kad, visi katalogai, išskyrus šakninį (/), priklauso kitam katalogui. Tokiu būdu, visi katalogai, išskyrus šakninį, yra pakatalogiai. Paprastumo dėlei mes juos vadiname katalogais.

Root kataloge yra trys, etc, tmp, ir usr vardais pavadinti pakatalogiai. Tokie vardai buvo parinkti todėl, kad jie yra trumpi ir lengvai surenkami klaviatūroje. Tačiau jie sunkiai ištariami. Todėl susitarta etc vadinti *etcetera,* tmp - *temp,* o usr - *user.* Pav., failo / et c/pas swd vardas tariamas *slash etcetera slash password.*

Root katalogas

Tai visos sistemos pagrindas. Paprastai jame talpinami tik kiti katalogai. Tačiau šaknyje būna vienas svarbus failas: programa, kuri vadinama Unix branduoliu (kernel). Berkeley Unix sistemose ši programa vadinama vmunix (virtual memory Unix). Unix sistemos menedžeris prisijungia prie sistemos root vardu. Dabar yra aiški šito vardo kilmė: *root* katalogas yra aukščiausiame failų sistemos lygyje.

Kelias	Turinys
/	'root' katalogas
/bin arba /sbin	minimaliam sistemos funkcionavimui reikalingos komandos
/dev	terminalus, diskus, modemus, etc. reprezentuojantys įtaisų failai
/etc	sistemos startavimo ir konfigūracijos failai
/lib	C kompiliatoriaus bibliotekos
/tmp	laikini failai, kurie išnyksta perkrovus sistemą
/sys	branduolio generavimo darbinė sritis, konfigūracijos failai (BSD)
/proc	visų vykdomų procesų atvaizdai (naujesnėse sistemose)
/stand	Stand-alone utilitos, diskų formatavimo priemonės, etc.
/usr/bin	vykdomieji failai
/usr/games	žaidimai ir kt. (daugelis jų nelabai įdomūs)
/usr/include	C programų h-failai
/usr/5bin	System V komandų analogai skirti BSD sistemoms
/usr/etc	sistemos palaikymo komandos
/usr/sbin	papildomos sistemos palaikymo komandos
/usr/lib	standartinių UNIX programų palaikymo failai
/usr/man	cOn-line manual' puslapiai
/var/adm	apskaitos failai, resusų naudojimo įrašai
/var/spool	printerių 'spooling' katalogai, UUCP, mail, etc.
/var/tmp	laikina sritis (filai neišnyksta po sistemos perkrovimo)
/usr/usb	Berkeley utilitos ir programos
/usr/local	lokali programinė įranga (jūsų instaliuoti paketai)
/usr/local/adm	lokalūs apskaitos ir šabloniniai failai
/usr/local/bin	lokalūs vykdomieji failai
/usr/local/etc	lokalūs sistemos konfiguravimo failai ir komandos
/usr/local/lib	lokalūs pagalbiniai failai
/usr/local/sbin	lokalios sistemos palaikymo komandos
/usr/local/src	/usr/local/* išeities kodas
/kernel	failai, kurie reikalingi branduolio užkrovimui (Solaris)

3 lentelė. Unix standartiniai katalogai ir jų turinys

/bin kataloge saugomos bazinės programos, iš kurių susideda pati Unix sistema. Čia galima rasti daugelį Unix komandų, t.y. failų, kurie vykdomi pagal atitinkamą komandą. Vardas *bin* gali būti traktuojamas dvejopai: daugelis failų yra dvejetainiai (binary); tai programų saugykla;

/dev kataloge yra specialūs failai, kurie reprezentuoja fizinius įtaisus. Paprastai vartotojai nenaudoja šių failų tiesiogiai. Tačiau norint taip daryti - Unix sistema yra atvira (vėliau bus pateiktas pavyzdys).

/etc katalogas skirtas sistemos administratoriui. Čia saugomi failai ir programos, reikalingi sistemos valdymui. Plačiausiai žinomas yra sistemos slaptažodžių failas *passwd.* Kitas žinomas failas yra *temcap* (kitose sistemose *tenninfo),* t.y. visų terminalų, kuriuos gali naudoti Unix, techninių apršymų duomenų bazė.

/lib kataloge yra programuotojų naudojamų programų biblioteka.

/lost+found katalogą naudoja speciali programa, kuri tikrina Unix failų sistemą. Kai ši programa randa failą, kuris nepriklauso jokiam katalogui (avarinis atvejis), tas failas talpinamas į *llost+found* katalogą. Tokiu būdu sistemos administratorius gali priimti tolimesnius sprendimus.

/sys kataloge yra taip vadinami sistemos išeities failai. Jie gali dominti sistemos administratorių ir programuotojus.

/tmp katalogas - tai laikina saugykla.Kiekvienas gali laikinai talpinti čia savo failus, tačiau laikas-nuo-laiko visi failai šalinami automatiškai. Paprastai čia talpinami laikini failai, kurie reikalingi programos vykdymui. Pav., naudojant *vi* redaktorių, redaguojamo failo kopija laikoma *l tmp* kataloge.

/usr kataloge yra visa eilė svarbių pakatalogių. Tai vienas iš svarbiausių šaknies pakatalogių. Jis pats suskirstytas į visą eilę pakatalogių.

/usr/bin katalogas, kaip ir jo bendravardis šaknyje, skirta vykdomųjų failų laikymui. /usr/dict - tai Unix žodyno naudojami failai. Faktiškai - tai žodžių sąrašas. /usr/games - tai žaidimų ir kitų atrakcijų katalogas.

/usr/include katalogas saugo taip vadinamus C programavimo kalbos 'include' (arba h) failus, kuriuos naudoja programuotojai įvairiose programose kompiliavimo metu. /usr/lib - tai analogiška /lib'ui biblioteka, svarbi tik programuotojams.

/usr/local - tai sistemos administratoriaus patogumui skirtas katalogas, kuriame gali būti saugomos vietinės reikšmės programos ir dokumentacija.

/usr/man katalogas turi visą eilę pakatalogių ir failų, kuriuos naudoja "Unix on-line manual", t.y. operatyvi [c]help' tipo dokumentacijos sistema.

/usr/spool - pakelės stotelės vaidmenį atliekantis katalogas. Čia laikomi duomenys, kurie buvo kur nors pasiusti. Pav., kai jūs spausdinate failą, tai tas failas, kartu su sistemine informacija laikomas šiame kataloge. Failas laikomas čia iki tol, kol galės būti atspausdintas. Lygiai taip pat čia patenka ir pasiusti elektroniniu paštu laiškai, kol jie iš tikrųjų 'palieka' kompiuterį.

/usr/src kataloge saugomas jūsų Unix sistemos išeities kodas. Jeigu jūsų Unix yra licenzijuotas (galima pirkti pačią Unix sistemą,arba jos licenziją), tai čia saugomas išeities kodas, leidžiantis "žvilgterėti j sistemos vidų", daryti pakeitimus ir perkompiliuoti sistemą.

Katalogo /dev specialių failų naudojimas: tty

Visi specialūs failai saugomi /dev kataloge. Pamatyti šių failų vardus leidžia komanda

ls /dev

Šiuos failus naudoja sisteminės programos, tačiau apie keletą jų verta žinoti plačiau.

Visi failai, kurie prasideda "tty", reprezentuoja terminalus. Pav., terminalą ttyOl reprezentuoja spec. failas /dev/ttyOl. Jeigu jūs rašysite informaciją į šį failą, ta informacija bus perduota į atitinkamo terminalo ekraną.

Užduotis.

Komanda who leidžia sužinoti, kokį terminalą naudoja jūsų kolega. Jus galite naudoti cp (copy) komandą, kuri kopijuoja vieną failą į kitą. Jeigu failescary yra pranešimas kolegai, tai tą pranešimą galite perduoti į kolegos terminaloekraną komanda cp scary /dev/ttyOl. Be to jūs galite sužinoti jūsų terminaląreprezentuojančio spec. failo vardą komanda tty. Rezultatas: ?

Visuomet galima naudoti /dev/tty failą, kaip jūsų terminalo sinonimą. Jei programa rašo duomenis į failą, tai vietoje failo vardo nurodžius /dev/tty, duomenys bus išvedami į jūsų ekraną.

Naudingiausias /dev katalogo failas yra null. Šis failas reprezentuoja neegzistuojantį įtaisą. Skaitant iš /dev/null įtaiso, 'nieko' nenuskaitoma; rašant į /dev/null, rašymas 'atšaukiamas'. Pav., jeigu programa updatef iles atnaujina duomenis tam tikruose failuose ir darbo protokolą išduoda į ekraną, tai pagal komandą updatef iles > /dev/null ši programa atliks savo 'darbą', tačiau protokolo išdavimas bus ignoruotas.

Namų katalogai

Unix sistemos sprendimas - skirti kiekvienam vartotojui *H ome Directory.* Tai su vartotojo *userid* asocijuotas ir vartotojo globojamas katalogas. Sistemos administratorius, registruodamas vartotoją, sukuria jam namų katalogą. Šio katalogo vardas saugomas sistemos slaptažodžių faile. Vartotojui prisijungus, jis automatiškai 'nuleidžiamas' į savo namų katalogą. Čia vartotojas turi teisę elgtis savo nuožiūra.

Tradiciškai namų katalogas kuriamos šakoje /usr, tačiau tai nėra taisyklė. Paprastai šio katalogo vardas sutampa su *'userid'.* Dažnai administratoriai vartotojus skirsto į grupes, pav., *undergrad, grad, guests, staff* ir pan. ir *home* katalogus kuria atitinkamose šakose. Vartotojui prisijungus, jo aplinkos kintamasis $HOME įgyja namų katalogo vardą. Be to, C-shell'as leidžia naudoti namų katalogo trumpinį ~ (tildė).

Užduotis.

Koks jūsų namų katalogo vardas vardas?

Pažiūrėkite $HOME kintamojo reikšmę (komanda echo $HOME), arba pažiūrėkite trumpinio ~ reikšmę (komanda echo ~).

12. Failų charakteristikos.

Šiame skyriuje jūs susipažinsite su UNIX komandomis, skirtomis failų bei katalogų charakteristikų peržiūrai, išmoksite keisti bei nustatyti įvairias failų prieinamumo galimybes. Su kiekvienu failu yra surišta eilė charakteristikų, kurias galime sužinoti naudodami l s komandą. Jos išvedamų į ekraną charakteristikų kiekį nusako komandos opcijos.

Pavyzdžiui, įvykdžius komandą:

ls -lgsF failo-vardas

matysime tokias charakteristikas:

4 –rwxr-xr-x 1 mindaugas mindaugas 54 2007-10-15 13.44 testing

1 2 3 4 5 6 7 8

1. Failo užimamų blokų skaičius
2. Failo prieinamumas – kas ir ką gali daryti su failu/katalogu
3. Failo ryšių kiekis
4. Failo savininko vardas
5. Failo savininko grupės vardas
6. Failo dydis baitais
7. Failo paskutinio modifikavimo data
8. Failo vardas

Antras laukas nusako failo tipą bei prieinamumą. Trečiame lauke yra nurodytas ryšių kiekis. Jis rodo, kiek vardų katalogų medyje yra surišta su tuo pačiu fiziniu failu. UNIX yra galimybė kreiptis į tą patį failą iš skirtingų vietų skirtingais vardais. Šie ryšiai yra sukuriami naudojant In komandą. Ketvirtas laukas rodo failo savininką. Kiekvienas UNIX failas turi savininką, juo yra vartotojas, sukūręs šį failą. Penktas laukas rodo grupę, kuriai priklauso failo savininkas. Septintame lauke yra nurodomas failo redagavimo laikas.

Failų prieinamumas

Paskutinės modifikacijos laikas. Šį laiką naudoja koordinatorius make, f ind komanda ir kt. Failo vardas, kuris nurodomas aštuntame lauke, gali turėti iki 14 simbolių (iki 255 BSD UNIX). UNIX nereikalauja, kad failo vardas būtinai turėtų išplėtimą, tačiau kai kurios programos, pavyzdžiui C kalbos kompiliatorius reikalauja, kad failų vardai turėtų plėtinį " . c". UNIX draudžia suteikti failui vardą "." arba "..", nes šie vardai atitinka einamąjį arba tėvo katalogą.

Failo tipą nusako antro lauko pirmas simbolis, tai yra:

Simbolis	Prasmė
-	paprastas(regular) failas
d	katalogas
b	blok-orientuotas specialus failas
c	simbolinio tipo specialus failas
1	simbolinis ryšys
p	Vamzdis (?) (pipe)
s	lizdas (socket)

Užduotis:

a) Įvykdykite komandą

```
ls -al ..
```

Kokius matote katalogus, kas jų savininkai, kokiai vartotojų grupei jie priklauso?

b) Įvykdykite komandą

```
ls -al /dev
```

Kokio tipo failai yra šiame kataloge?

c) Įvykdykite komandą

```
ls -al /bin
```

Kokio tipo ir dydžio failus matote?

Sekantys devyni simboliai charakterizuoja failo prieinamumą. Sistemos administratorius turi visas prieinamumo galimybes peržiūrėti vartotojų failus bei katalogus, nežiūrint į tai, kokios jos yra užduotos.

Lentelėje pateiktame pavyzdyje nurodytos tokios failo prieinamumo reikšmės: rwxrw-rw-

Šie prieinamumo simboliai yra skirstomi į tris grupes po tris simbolius ir nusako priėjimo prie failo galimybes: pirmi trys simboliai galimybes vartotojui - failo savininkui, antri trys - vartotojams, kurie priklauso tai pačiai grupei kaip ir failo savininkas, paskutiniai trys -likusiems vartotojams. Čia r raidė žymi galimybę failą skaityti, w - failo rašymo galimybę, x - vykdymo galimybę. Brūkšnelis rodo, kad atitinkamos galimybės nėra.

Pavyzdžiui:

rwxrw-r-- prieinamumo reikšmės žymi, kad failo savininkas turi teisę rašyti skaityti ir vykdyti šį failą, grupės vartotojai gali skaityti ir rašyti, o visi kiti -tik skaityti.

Visos šios galimybės yra kiek skirtingai traktuojamos priklausomai nuo failo tipo.

Paprastam failui skaitymo galimybė reiškia, kad procesas gali failą skaityti, rašymo galimybė, - kad procesas gali keisti failo turinį, vykdymo galimybė, - kad procesas gali šį failą vykdyti.

Katalogui skaitymo galimybė reiškia, kad procesas gali skaityti katalogą(tai yra perskaityti sąrašą failų, esančių tame kataloge). Rašymo galimybė reiškia, kad procesas gali išmesti failus iš katalogo arba juos įkelti į katalogą. Vykdymo galimybė reiškia, kad procesas gali prieiti prie failų esančių kataloge arba jo pakatologiuose.

Specialiam failui skaitymo galimybė reiškia, kad procesas gali skaityti iš failo, naudodamas kreipinį read(). Analogiškai rašymo galimybė reiškia, kad procesas gali rašyti į failą, naudodamas kreipinį write(). Vykdymo galimybė neturi prasmės.

Užduotis:

l s komanda išvedė tokius rezultatus:

drwxrw-r— 2 tomas userl 512 Jan 30 05:14 kat1

drwxr—r— 2 petras user2 512 Jan 30 08:14 kat2

Vartotojas Domas priklauso grupei userl, o vartotojas Elijus grupei user3. Kokius veiksmus gali atlikti šie vartotojai kataloguose kati bei kat2?

Užduotis:

f ile komanda atlieka failų identifikavimą. Duokite komandą

file *

Paskaitykite komandos file *Manual* puslapį.

Realus bei efektyvus identifikatoriai

Su kiekvienu vykdomu procesu yra surišami keturi identifikatoriai: realus vartotojo identifikatorius efektyvus vartotojo identifikatorius realus grupės identifikatorius efektyvus grupės identifikatorius.

Su šiais identifikatoriais irgi yra surištos tam tikros prieinamumo galimybės. Vartotojui prisijungiant prie sistemos startuoja shell‘o procesas, kuris turi nustatytus vartotojo realų ir efektyvų identifikatorius, sutampančius su vartotojo identifikatoriumi, realų ir efektyvų grupės identifikatorius, sutampančius su vartotojo grupės identifikatoriumi. Kadangi failo prieinamumo reikšmės gali būti skirtingos vartotojui, grupei bei kitiems vartotojams, tai juos taikant yra atliekamas tikrinimas:

- jei proceso efektyvus vartotojo identifikatorius yra toks pats kaip failo savininko identifikatorius, tai yra pritaikomos vartotojui nustatytos prieinamumo galimybės,
- jei proceso efektyvus vartotojo identifikatorius skiriasi nuo failo savininko identifikatoriaus, bet efektyvus grupės identifikatorius sutampa su failo grupės identifikatoriumi, tai pritaikomos grupės prieinamumo galimybės,
- likusiais atvejais taikomos kitiems vartotojams skirtos failo prieinamumo galimybės.

Visos šios prieinamumo galimybės leidžia apsaugoti failus nuo nesankcionuoto priėjimo bei naudotis bendrais failais, jei vartotojai priklauso tai pačiai vartotojų grupei.

Vykdomiesiems failams UNIX sistemoje gali būti nustatytas specialus bitas, s (the sėt user/group id bit), liepiantis nustatyti vartotojo identifikatorių arba grupės identifikatorių. Jis atžymimas s raide vartotojo arba grupės laukuose vietoje simbolio x. Jei failui yra nustatytas šis bitas, tai failo vykdymo metu efektyvus vartotojo identifikatorius arba efektyvus grupės identifikatorius yra keičiamas į vykdomo failo savininko identifikatorių arba jo grupės identifikatorių, o tai suteikia vartotojui šio failo vykdymo metu tokias pačias teises kaip ir failo savininkui. Tai įgalina vartotoją keisti įrašą slaptažodžių faile, nors šio failo savininkas yra root.

Šie bitai kaip ir failo prieinamumo galimybės failams gali būti nustatomi naudojant chmod komandą.

Failo prieinamumo nustatymas

Naujai kuriamiems failams prieinamumo galimybės nustatomos pagal nutylėjimą, tačiau vartotojas gali jas užduoti ir pats. Failams prieinamumo galimybės gali būti nustatomos ir keičiamos šiomis komandomis:

Komanda	Paskirtis
chmod	Keičia failo prieinamumo galimybes
umask	Nustato prieinamumo galimybes naujai kuriamam failui
chgrp	Keičia grupę, kuriai priklauso failas
chown	Keičia failo savininką

Komanda chmod

Failo prieinamumo galimybes galima keisti komanda

chmod -R *pakeitimas {failo-vardas}*

R raktas rodo, kad nustatymas galios rekursyviai, t.y. visuose katalogo pakatalogiuose. Lauke *{failo vardas}* gali būti nurodomi failai bei katalogai, o *"pakeitimu"* yra nurodoma kokiems vartotojams ir kas yra nustatoma. Vartotojas žymimas vienu simboliu:

u	savininkas
g	grupė
o	kiti
a	visi

Šis simbolis skelbia, kokiems vartotojams yra nustatomas failo ar katalogo prieinamumas. Pats prieinamumas nusakomas raide:

r	skaityti
w	rašyti
X	vykdyti
s	nustatyti vartotoj o ar grupės identifikatorių

Prieinamumas gali būti nurodomi tiek aukščiau išvardytomis raidėmis, tiek ir skaičiais: jei atitinkama galimybė leidžiama yra l, o jei ne - 0. Vienetuku-nuliukų seka vėliau gali būti užrašoma kaip aštuntainis skaičius.

Pavyzdžiui: Simbolių keitimas skaičiais

rwx r-- r-x simbolinis

111 100 101

7 4 5

Komandos chown bei chgrp

Naujų vartotojų grupių sukūrimą UNIX sistemoje vykdo sistemos administratorius. Sukūrus naują grupę, vartotojai gali būti paskelbiami šios grupės nariais. Vartotojas gali priklausyti kelioms grupėms.

Norint sužinoti, kokių grupių nariu yra vartotojas, jis gali panaudoti komandą groups [*vartotojo-identifikatorius*]

Nenurodžius vartotojo-identifikatoriaus yra išvedamas grupių sąrašas vartotojui, įvykdžiusiam šią komandą.

Vartotojas gali pakeisti grupę, kuriai priklauso failas, įvykdydamas komandą chgrp -R *grupės-identifikatorius {failo-vardas}*

vartotojui leidžiama pakeisti grupę tik tiems failams, kurių savininku yra jis pats, o sistemos administratoriui leidžiama keisti bet kurio failo grupę. Komandoje gali būti nurodytas failų sąrašas.-R raktas liepia keisti grupę rekursyviai visiems katalogo failams. Failo savininką gali keisti sistemos administratorius, naudodamas komandą

chown -R *naujas-vartotojas {failo-vardas}.*

Vietoje failo gali būti nurodomas ir katalogas. Kurdamas namų katalogą naujam vartotojui, sistemos administratorius visuomet privalo keisti šio katalogo savininką bei grupę, nes, administratoriui sukūrus naują katalogą, jis pradžioje ir priklausys administratoriui bei turės administratoriaus grupės identifikatorių. Tas pats bus ir su failais, kuriuos administratorius turi įkelti į vartotojo katalogą: . cshrc . login .logout. Taigi, administratorius privalo vykdyti chown bei chgrp komandas su

opcijomis -R, kad vartotojas butų paskelbtas katalogo bei failų savininku, o grupė nustatyta tokia, kokia yra vartotojo grupė.

Jei vartotojas priklauso kelioms grupėms, tai jis gali pakeisti grupę, vykdydamas komandą

newgrp *grupės-identifikatorius*

Šiuo atveju bus sukuriamas naujas shell'o procesas, kuris turės naują grupės identifikatorių, o pradinis shelPo procesas lauks šio naujo proceso pabaigos.

13. Failų sistemos komandos

UNIX katalogų ir kelių klasifikacija

Katalogai (Directories): UNIX katalogai apjungti į medžio tipo struktūrą. Kataloguose saugomi žemesnio lygio katalogai ir failai.

Šakninis(Root) katalogas:yra medžio viršuje ir žymimas / simboliu.

Namų(Home) katalogas: jis dar vadinamas login katalogu. Sukuriamas kiekvienam sistemos vartotojui ir į jį patenkama automatiškai, prisijungus prie sistemos.

Absoliutus kelias iki einamojo katalogo

pwd (print working directory)

pwd komanda parodo pilną kelią nuo šaknies iki einamojo katalogo. Pasitikrinkite koks jūsų einamasis katalogas. Tai viena iš nedaugelio komandų, neturinti raktų (opcijų). Be to ši komanda (ji yra vidinė shell'o komanda) galioja visuose plačiai naudojamuose shell'uose csh, tcsh, sh, ksh ir pan.

Naujų katalogų sukūrimas

Komanda:

mkdir *katalogo_yardas*

Ši komanda einamajame kataloge sukuria katalogą.

Užduotys:

Namų kataloge sukurkite katalogus: Iab2, Iab3, Iab4 , Iab5.

Koks sukurtų katalogų prieinamumas, kas jų savininkas, kokia grupė? p.s. pagal nutylėjimą (sitemos nustatytas) prieinamumas katalogams yra 777, o failams 666. Iš šių reikšmių yra minusuojama umask reikšmė ir tokiu būdu gaunama naujai sukurtų objektų prieinamumo reikšmė.

Uždrauskite sukurtų katalogų Iab2, Iab3, Iab4 , Iab5 prieinamumą kitiems vartotojams, išskyrus jūsų grupę.

Katalogo turinio peržiūrėjimas naudojant įvairias opcijas

ls *katalogas*

Ši komanda leidžia, naudojant įvairius raktus (opcijas) gauti platesnę arba siauresnę informaciją apie failus bei katalogus, esančius nurodytame kataloge.

Užduotis:

Naudodami man susipažinkite su Iš komanda ir jos pateikiamais rezultatais. ls komanda peržiūrėkite einamojo katalogo turinį. Panaudokite įvairias opcijas.

Susipažinkite su komanda dir.

Žiūrėti failų tipus ir i-node numerius [ls -Fi]

Žiūrėti kiekvieną pakatalogį ir jų turinius rekursiškai [ls -R /kernai]

Perėjimas į kitą katalogą

cd *katalogas*

cd komanda naudojama perėjimui į kitą katalogą. Pavyzdžiui įvykdžius

cd labi labi taps einamuoju katalogu,

cd .. perkels medžiu aukštyn,

cd ../.. perkels medžiu aukštyn per du lygius,

cd grąžins jus į *home* katalogą,

cd / perkels į šakninį katalogą.

Šioje komandoje esantys du taškai žymi perėjimą katalogų medžiu vienu lygiu aukštyn, vienas taškas naudojamas einamojo katalogo nurodymui, ~ ženklas ir po jo sekantis vartotojo *login vardas* leidžia pereiti į nurodyto vartotojo namų katalogą.

Užduotis:

Pereikite į katalogą Iab2

Įvykdykite komandas

pwd

ls -a

dirs Kokia tai komanda, vidinė ar išorinė?

dirs -l

Failų perkėlimas į kitą katalogą

mv komanda naudojama perkeliant failą arba pakeičiant katalogo(failo) vardą. Perkeliant failą iš einamojo katalogo į pakatalogį vykdoma komanda:

mv *failas pakatologis/*

Šiuo atveju failas bus perkeltas tuo pačiu vardu.Perkeliant galima keisti failo vardą:

mv *failasl pakatologis/ failas2*

Katalogas, į kurį vykdomas failo perkėlimas gali būti nurodomas nurodant absoliutų kelio vardą.

Užduotis:

Perkelkite failus iš savo namų katalogo į katalogą Iab2

cat komandos teikiamos galimybės

Sistemoje UNIX naują failą galima sukurti įvairias būdais, vienas paprastesnių yra jo įvedimas iš klaviatūros prieš tai surinkus cat komandą:

cat > *naujo-failo-vardas*

Po to yra įvedami failo įrašai. Įvedimo pabaiga nurodoma ^ D klavišais.

Užduotis:

Pereikite į katalogą lab3.

Komanda cat sukurkite naują failą vardu pavyzdi, kuriame talpinsime 3 eilutes žemiau pateikiamo teksto:

```
cat > pavyzdis
tai pirma eilute
tai antra eilute
ir trečia eilute
^D
```

cat komanda dažnai naudojama tekstinių failų peržiurai. Tai atliekama surinkus komandą

```
cat failo-vardas
```

Be to, naudojant cat komandą galima apjungti failus ir jų turinys parodomas ekrane. Failų apjungimas yra tikroji šios komandos paskirtis.

Užduotis:

Sukurkite failą pavyzd2, į kurį įrašykite sekantį tekstą:

```
O tai ketvirtoji eilutė.
^D
```

Atlikite failų apjungimą, naudodami komandą:

```
cat pavyzd1 pavyzd2
```

Informacinių srautų perskirstymas

UNIX sistemoje komandos pagal nutylėjimą duomenis ima iš klaviatūros (standartinis įvedimo failas), o komandų įvykdymo rezultatai yra nukreipiami į ekraną (standartinis išvedimo failas). Komandų vykdymo rezultatai gali būti nukreipiami į failą, panaudojant simbolį >. Tokią konstrukciją jau panaudojome kurdami failus pavyzdi pavyzd2. Dabar nukreipkime failų pavyzdi ir pavyzd2 apjungimo rezultatą į failą catout:

```
cat pavyzdi pavyzd2 > catout
```

Pastaruoju atveju komandos įvykdymo rezultatų ekrane nematysim, bus sukurtas naujas failas catout. Jį galima peržiūrėti komanda:

```
cat catout
```

Pastaba: Jei nukreipiate standartinį išvedimą į failą

```
komanda > failo-vardas
```

tai:

komandos įvykdymo rezultatų ekrane nebus, jie bus f aile/a*70-vardas;

jei tokio failo *failo-vardas* dar nebuvo, jis bus sukurtas;

jei failas *failo-vardas* jau buvo, jo turinys bus išgadintas.

Simbolis > nukreipia standartinį išvedimą (terminalo ekranas) į nurodytą failą. Atitinkamai norint įvesti duomenis ne iš klaviatūros, o imti iš failo, yra naudojamas simbolis <.

Todėl komanda

komanda < failasl > failas2 žymės tai, kad duomenys bus imami iš failo *failasl,* o rezultatai talpinami faile *failas2.*

Failo turinio peržiūrėjimo komandos

Tai: more, head ir tail komandos.

more komanda, taip pat kaip ir cat komanda, parodo failo turinį ekrane. Ji naudotina ilgų failų peržiūrai, kadangi more komanda išveda informaciją į ekraną puslapiais:

more *failo-vardas*

Pereiti prie kito puslapio galima spaudžiant tarpo klavišą. Šią komandą galima naudoti kitų komandų išvedamų rezultatų peržiūrai.

Pavyzdžiui:

ls -al | more

Panaudojus komandą head galima pamatyti pirmas 10 failo eilučių:

head *failo-vardas*

Panaudojus komandą tail galima pamatyti paskutines 10 failo eilučių:

tail *failo-vardas*

Užduotis:

Peržiūrėkite turinį su komanda cat ir more.

Panaudokite komandas head ir tail katalogo /bin turinio peržiūrėjimui.

Failų kopijavimas

Failo kopijavimui naudojama komanda cp. Įvykdžius komandą:

cp failasl failas2 bus gauta failo *failasl* kopija.

Aplamai reikėtų prisiminti bendrą katalogų kopijavimo komandos formą

cp -r *katalogasl katalogas2*

Komanda kopijuoja *katalogasl* ir visa kas j ame yra į katalogą *katalogas2.* Pastarasis katalogas sukuriamas, jeigu jo nėra. Jeigu *katalogas2* yra, tai jame bus sukurtas pakatalogis *katalogasl.*

Vietoje kopijavimo komandų dažnai naudojama failų ir katalogų surišimo komanda ln.

Failų ir katalogų naikinimas

rmdir komanda pašalina tuščius katalogus iš einamojo katalogo,

rm *-opcijos failas*

ir jos opcijos leidžia naikinti failus. Reikia atsiminti, kad atstatyti išmestus katalogus arba failus galima tik tuo atveju, jei sistemos administratorius yra atlikęs atminties išsaugojimą.
Ši komanda naikina failą ir jo turinį.

Failo spausdinimas

Norint failą spausdinti popieriuje, naudojama komanda lpr (lp) :

lpr *failo-vardas*

Spausdinimas galimas tik esant pajungtam spausdintuvui.

lpq komanda naudojama printerio eilės peržiūrėjimui: lpq

Komanda lprm naudojama norint išmesti tam tikrą užduotį iš spausdinimo eilės. Užduoties numerį galima sužinoti komanda lpq, o po to naudoti komandą

lprm *uzduoties-nr*

wc komanda

Šios komandos pagalba galima suskaičiuoti eilučių, žodžių ir simbolių kiekį viename arba keliuose failuose:

wc *failas l failas2*

Kontroliniai klausimai.

Kaip galima sužinoti, kokie katalogai yra einamajame kataloge?
Kaip sužinoti, ar failas yra vykdomasis?
Kaip atžymimi specialūs failai, ryšiai (link)?
Kaip peržiūrėti kito katalogo turinį, nekeičiant einamojo katalogo?
Kaip perkelti visus failus į kitą katalogą?
Kaip perskirstyti įvedimo/išvedimo duomenis?
Kaip įrašyti paskutines failo eilutes į naują failą?
Kaip išnaikinti katalogą su visu jo turiniu?