Operacinės sistemos

N. Sarafinienė

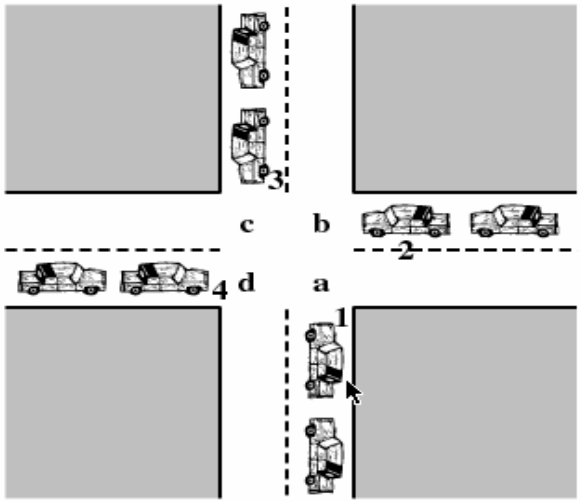
2013m.

Kalbėsime

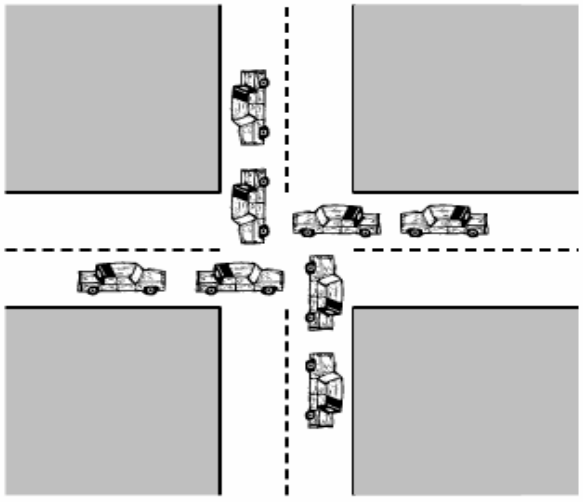
- Komunikacija tarp procesų UNIX sistemoje
- Mirties taškas
  - Jo susidarymo sąlygos
  - Mirties taško prevencija
  - Mirties taško vengimas
  - Mirties taško nustatymas
  - Išėjimas iš mirties taško

Mirties taškas ir jo išvengimo būdai

- Mirties taškas sistemoje - tai tokia situacija, kai ilgam laikui yra užblokuojama grupė procesų, kurie varžosi dėl tų pačių sistemos išteklių arba komunikuoja su kitais, užblokuotais procesais.
- Mirties taškas kyla kai du ar daugiau procesų turi konfliktuojančius poreikius.
- Dažniausiai nėra bendro sprendinio, kuris nusakytų kaip elgtis mirties taško atveju. Kai kurių operacinių sistemų kūrėjai skaito, kad jų sistemose mirties taškas negali niekad susidaryti. Pavyzdžiu gali būti operacinė sistema - Unix SVR4.



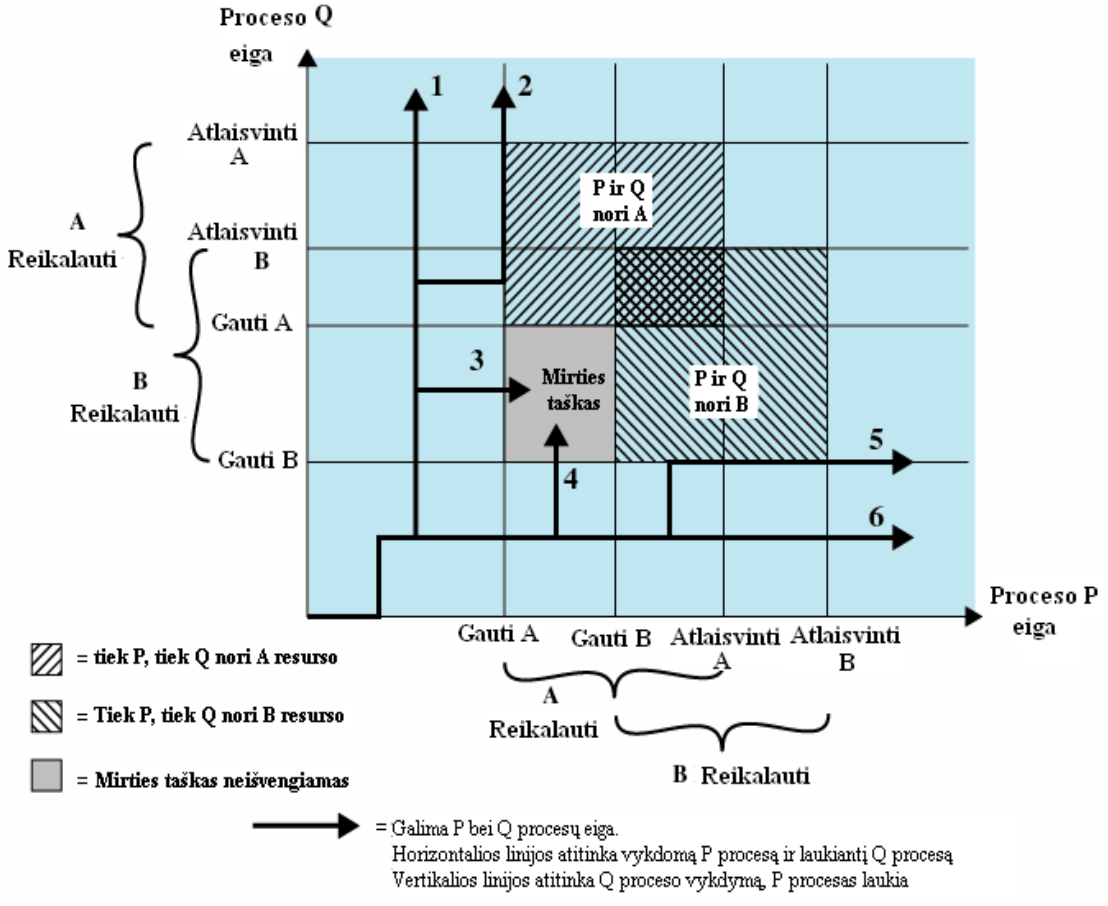
a) Galimas mirties taško susidarymas



b) Mirties taško situacija

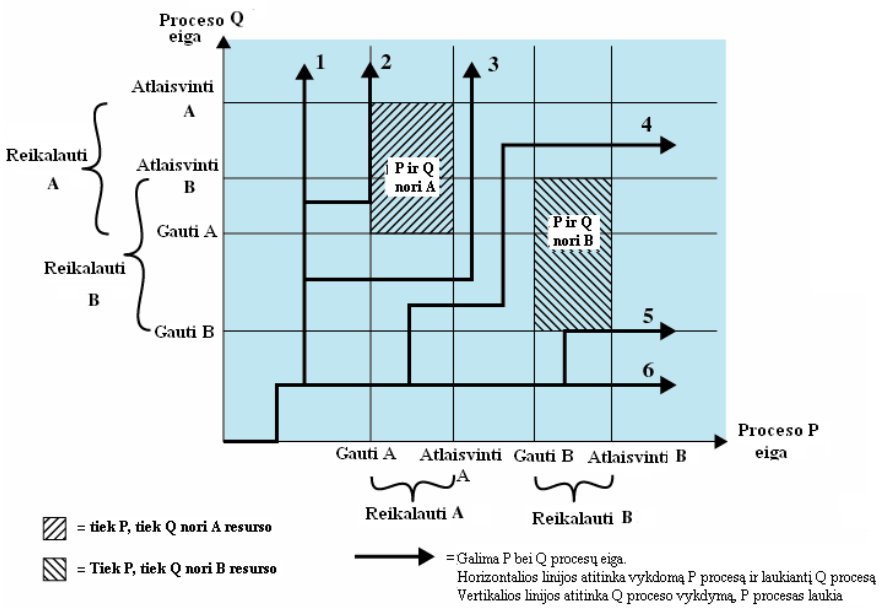
Pavyzdys

- Tarkim: turime du procesus – P ir Q
- P procesas pradžioje nori gauti resursą A, o po kurio laiko - resursą B.
- Procesas Q- atvirkščiai, pradžioje nori gauti resursą B, o po to resursą A.
-



Pavyzdys

- Tačiau mirties taškas gali ir nesusidaryti, net ir šiuo atveju, kai abu procesai reikalauja tų pačių resursų, jei vienas iš procesų nereikalauja abiejų resursų vienu metu, kaip kad parodyta paveiksle



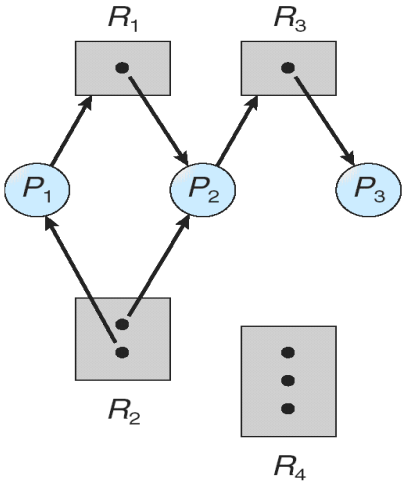
Mirties taško susidarymo sąlygos

Mirties taškas sistemoje tampa galimu, jei sistemoje susidaro šios sąlygos:

- **Tarpusavio išskirtinumo**
- **Turėjimo ir laukimo**
- **Neperėmimo**
- **Ciklinio laukimo sąlyga**

Resursų priskyrimas procesams

- procesų resursų poreikiai gali būti atvaizduojami resursų priskyrimo grafais .
- Resursas gali turėti keletą identiškų elementų (du spausdintuvai), grafe tai atžymima keliais taškais **R** tipo dėžutėje (**R2, R4** resursas).
- Jei procesui resursas yra priskirtas, tai rodyklė nukreipta nuo resurso į procesą (**R1** yra priskirtas procesui **P2**).
- Jei procesas reikalauja resurso, tai rodyklė nukreipta nuo proceso į resursą (**P1** reikalauja **R1**).

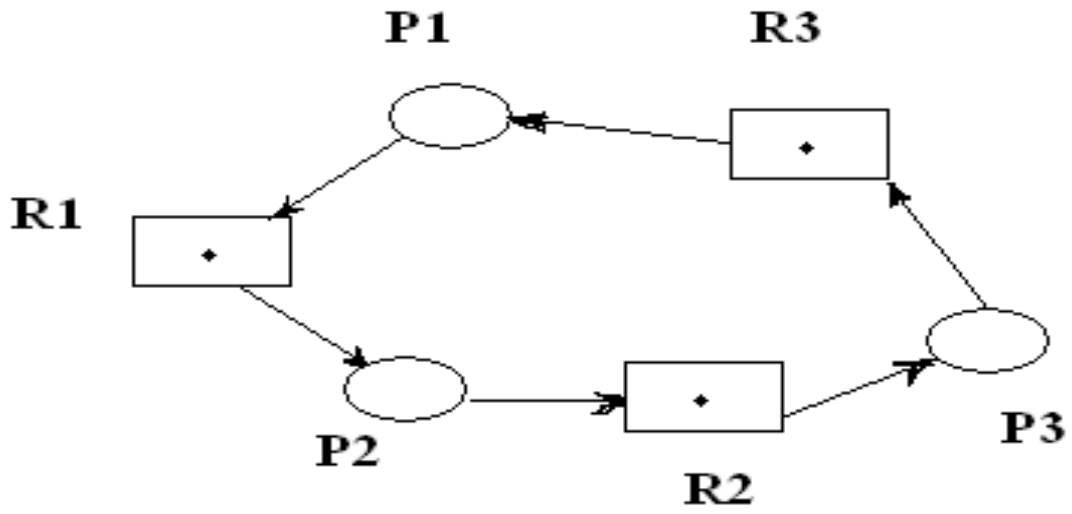


R_1, R_2, R_3, R_4 - sistemos resursai
P_1, P_2, P_3 - procesai

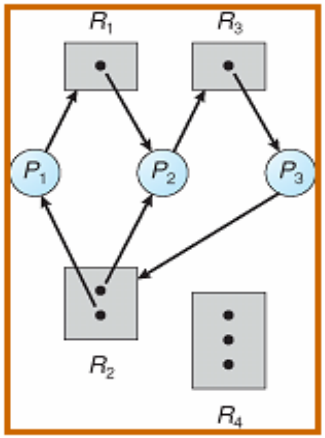
Mirties taško situacija

Sistema yra mirties taško situacijoje, nes kiekvienas procesas yra užsiėmęs resursą, kurio prašo kitas procesas ir joks procesas nenori atiduoti resurso, kurį jis valdo.

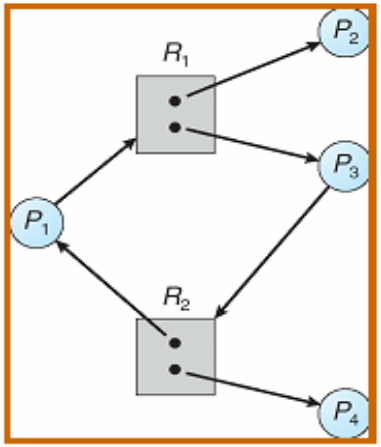
Ciklinio laukimo salyga



Resursų priskyrimo grafų pavyzdžiai



a) Resursų priskyrimo grafas su mirties tašku



b) Grafas su ciklu bet be mirties taško

Mirties taškas ir sprendimai

- Egzistuoja 4 galimybės
  - Sutrukdyti mirties taškui susidaryti (*mirties taško prevencija*)
  - *Vengti mirties taško* susidarymo
  - Nustatyti, kad mirties taškas susidarė
  - Išeiti iš susidariusio mirties taško

Mirties taško prevencija (sutrukdymas)

- Mirties taško sutrukdymui operacinėse sistemose yra naudojami *tiesioginiai* arba *netiesioginiai* būdai:
  - **Netiesioginiams** mirties taško sutrukdymo metodams būdinga tai, kad jie neleidžia susidaryti vienai iš trijų paminėtų sąlygų.
  - **Tiesioginiai** metodai sutrukdantys mirties taško susidarymui paprastai neleidžia susidaryti cikliniam laukimui.

Netiesioginiai metodai

Netiesioginiai metodai yra orientuoti į pirmas tris sąlygas

1. Tarpusavio išskirtinumo sąlyga
   - Šios sąlygos egzistavimo dažnai negalima suardyti,
2. Turėjimo ir laukimo sąlyga
   - Šią sąlygą galima suardyti, reikalaujant kad procesas visų jam reikalingų resursų užprašytų iš karto .
   - Tokio metodo minusai:
     - Gali gautis **badavimo** situacija.
     - **žemas resursų panaudojimas**.
     - procesas turi žinoti, kokių resursų ir kiek jų jam reikės.

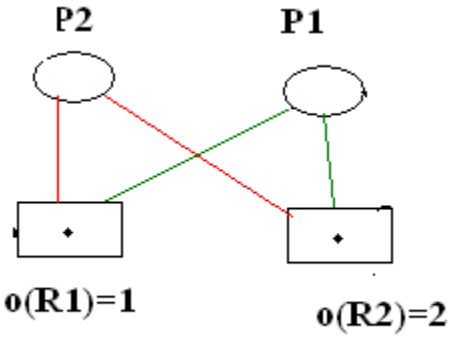
Netiesioginiai metodai

Neatėmimo sąlyga.

- Ši sąlyga gali būti sutrukdoma keliais būdais. Metodo esmė yra ta, kad iš proceso yra atimami jam anksčiau išskirti resursai.
  - Jei procesas turi kažkurį resursą ir jam negali būti tuojau pat paskirti kiti jam tuo momentu reikalingi resursai, tai jo turėti *resursai iš jo yra atimami*.
  - Bet, jei procesas turi atlaisvinti resursą, kurį jis naudoja, šio resurso būvis turi būti *išsaugomas*, kad vėliau būtų galima pratęsti (atstatyti) proceso vykdymą.
  - .

Tiesioginiai metodai siekiant išvengti mirties taško susidarymo

- Pateiksime protokolą, kurio laikantis galima ciklinio laukimo prevencija:
- Nustatoma griežta tiesinio surikiavimo eilė **O()** įvairiems resursų tipams. Pavyzdžiui:
  - **R1**:magnetinėms juostoms **O(R1) =2** (eiliškumas 2)
  - **R2**:diskiniams įrenginiams **O(R2)=4**.
  - **R3**:spausdintuvams **O(R3)=7**.
- Procesui gali reikėti įvairių resursų tipų. Viena užklausa jis gali užsiprašyti kelių vieno kažkurio resurso, tarkim **Ri**, vienetų.
  - Yra reikalaujama, kad skirtingų resursų užprašymas būtų atliekamas atskira užklausa.
- Procesas, kuriam išskirtas resursas **Ri** gali užprašyti ir gauti kelis resurso **Rj** vienetus tik tuo atveju, jei **O(Rj)>O(Ri)**.
- Šio reikalavimo prisilaikant **nesusidarys** ciklinis laukimas.



Mirties taško prevencija: Išvados

- Reikia suardyti vieną iš sąlygų, kuri sąlygoja mirties taško susidarymą arba reikia naudoti protokolą, kuris apsaugo nuo ciklinio laukimo susidarymo

- *Visa tai gali iššaukti neefektyvų resursų panaudojimą bei neefektyvų procesų vykdymą*

Mirties taško vengimas

- Šiuo atveju leidžiama įvykti pirmoms trims sąlygoms, tačiau vykdomas protingas resursų išskyrimas, kuriam esant mirties taškas niekad nebus pasiektas.
- Taikant šį metodą yra leidžiama didesnė konkurencija dėl resursų nei prevencijos atveju.
- Taikomi du būdai:
  - **Procesas nepradedamas**, jei jo resursų užklausa gali vesti į mirties taško susidarymą.
  - **Resursų padidinimo užklausa nevykdoma**, jei toks išskyrimas ves prie mirties taško susidarymo.
- Abiem šiais atvejais iš anksto turi būti nusakomi maksimalūs proceso poreikiai resursams.

Resursai ir proceso poreikis jiems

- Sistemos resursai yra suskirstomi į tam tikrus resursų tipus. Sistemoje paprastai yra tam tikras, ribotas kiekvieno resurso kiekis.

- **Proceso pradėjimo (inicijavimo) atmetimas.**
  - Lai **C(k,i)** žymi tai, kokio **max** **i**-tojo resurso kiekio nori **k**-tasis procesas.
  - **k**-tasis procesas turi nurodyti savo **poreikius** visiems **i**-tiems resursams - **C(k,i)**.
  - Lai **U(i)** bus bendras **nepareikalautas i-to resurso kiekis**.
  - Naujas, **n**-tas procesas yra priimamas į sistemą tik tuo atveju, jei **C(n,i)<U(i)** visiems resursams **i**.
- Ši politika garantuos tai, kad mirties taško bus visad išvengta, kadangi procesas yra priimamas tik tuo atveju, jei jo užklausa gali būti pilnai patenkinama (nežiūrint to, kokia tvarka procesas pareikalaus tų resursų).
- Tai suboptimali strategija kadangi ji įvertina blogiausią situaciją, tai yra tokią situaciją, kada visi procesai pareikalaus maksimalių resursų kiekių vienu metu.

Bankininko algoritmas

- Procesai yra kažkuo panašūs į vartotojus, norinčius pasiskolinti pinigų (resursų) banke. Bankininkas turėtų neduoti paskolos, jei jis negali patenkinti visų vartotojų poreikių.
  - Bet kuriuo momentu sistemos būvis gali būti apibrėžiamas **R(i)**, **C(j,i)** reikšmėmis, kur **i** žymi **R** resurso tipą, o **j** procesą.
  - Matricos **A** elementas **A(j,i)** parodo, koks **i** –tojo resurso kiekis jau yra skirtas **j**-tajam procesui. (visoms **j,i** reikšmėms).
  - Sumarinis esamas laisvas **i** –tojo resurso kiekis yra nusakomas vektoriaus **V(i)**-taja komponente:
  - Matricos **N** elementai **N(j,i)** nusako, kiek **i**-to resurso reikia tam, kad **j**-tasis procesas galėtų baigti įvykdyti savo užduotį.

Bankininko algoritmas

- Sprendžiant, ar gali būti patenkinama proceso užklausa resursui, bankininko algoritmas testuoja, ar užklausos patenkinimo rezultate sistemos būvis gausis **saugus**.
  - jei skyrus resursą procesui rezultate gausis saugus būvis, tai užklausa yra tenkinama,
  - priešingu atveju ji nėra tenkinama.
- **Būvis yra laikomas saugiu,** jei galima sudaryti tokią procesų seką **{P1...Pn}**, kuriai esant bus galima kiekvienam iš sekoje esančių procesų priskirti visus to proceso įvykdymui reikalingus resursus. Esant saugiam būviui visada visi procesai bus pilnai įvykdyti.

Saugaus būvio nustatymo algoritmas

Inicializacija:

Visi procesai skelbiami nebaigtais.

Darbiniam vektoriui **W(i)** priskiriame esamų resursų kiekius

W(i)=V(i) visiems **i**;

Kartojama:

Ieškomas toks neužbaigtas procesas **j**, kuriam :

N(j,i)<=W(i), visiems **i**.

- jei tokio **j** nera - pereiti i **EXIT**
- jei toks **j** yra "užbaigti" šį procesą ir grąžinti to proceso užimtus resursus - skelbiant juos laisvais:

W(i)=W(i)+A(j,i) visiems **i**.

Grįžti į kartojimą.

EXIT: jei visi procesai tapo "užbaigtais", tai būvis laikomas saugiu,

priešingu atveju - nesaugiu.

Bankininko algoritmas

- Lai **Q(j,i)** tai ***i–tojo*** resurso kiekis, kurio tam tikru momentu užsiprašo ***j-tasis*** procesas.
- **Algoritmo žingsniai**:
  - Jei **Q(j,i)<=N(j,i)** visiems **i**, tai einama prie sekančio žingsnio, priešingu atveju gaunama klaidos sąlyga (**j**-tojo proceso resursų pareikalavimas viršytas).
  - Jei **Q(j,i)<=V(i)** visiems **i** tai eiti prie sekančio žingsnio, priešingu atveju laukti, nes esamų laisvų resursų kol kas nepakanka.
  - Pretenduojama į tai, kad resurso poreikiai bus tenkinami ir nustatomas naujas resurso išskyrimo būvis:
    - **V(i)=V(i)-Q(j,i)** visiems **i**
    - **A(j,i)=A(j,i)+Q(j,i)** visiems **i**
    - **N(j,i)=N(j,i)-Q(j,i)** visiems **i**
  - Jei šio priskyrimo rezultate būvis gaunasi saugus, tai iš tikrųjų yra įvykdomas resurso **Q(j,i)** išskyrimas procesui **j**. Priešingu atveju **j** procesas turės laukti resurso **Q(j,i)** išskyrimo ir išsaugomas senas būvis.

Bankininko algoritmo pavyzdys

C(j,i)

	Reikia max		
	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

A(j,i)

Jau yra išskirta		
R1	R2	R3
1	0	0
5	1	1
2	1	1
0	0	2

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

Matricos **C** elementai **C(j,i)** nusako, kiek **i**-to resurso maksimaliai reikia **j**-tajam procesui.

Matricos **A** elementai **A(j,i)** parodo, koks **i** –tojo resurso kiekis jau yra skirtas **j**-tajam procesui (visoms j,i reikšmėms).

Matricos **N** elementai **N(j,i)** nusako, kiek **i**-to resurso reikia tam, kad **j**-tasis procesas galėtų baigti įvykdyti savo užduotį.

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

A(j,i)

Jau yra išskirta		
R1	R2	R3
1	0	0
5	1	1
2	1	1
0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
1	1	3

Esamas laisvas **i –**tojo resurso kiekis yra nusakomas vektoriaus **V(i)**-taja komponente.

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
1	1	3

Lai **Q(j,i)** tai **i–tojo** resurso kiekis, kurio tam tikru momentu užsiprašo **j-tasis** procesas.

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
1	1	3

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Tarkime P2 procesas paprašė vieno vieneto R1 ir vieno vieneto R3 resurso.

Q(P2) = (1, 0, 1)

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
1	1	3

Algoritmo žingsniai:

- Tikriname sąlygą **Q(P2) <= N(P2)**

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Bankininko algoritmas

N(j,i)

	Dar reikia			Jau yra išskirta		
	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0
P2	1	0	2	5	1	1
P3	1	0	3	2	1	1
P4	4	2	0	0	0	2

A(j,i)

V(i)

Laisvas kiekis		
R1	R2	R3
1	1	3

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Tikriname sąlygą **Q(P2) <= N(P2)**

.	Q(P2)		N(P2)
R1	1	=	1
R2	0	=	0
R3	1	<	2

Sąlyga tenkinama, **klaidos nėra**.
Einame į kitą **žingsnį.**

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
1	1	3

Algoritmo žingsniai:

- Tikriname sąlygą **Q(P2, i) <= V(i)** visiems i.

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Bankininko algoritmas

N(j,i)

	Dar reikia			Jau yra išskirta		
	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0
P2	1	0	2	5	1	1
P3	1	0	3	2	1	1
P4	4	2	0	0	0	2

A(j,i)

V(i)

Laisvas kiekis		
R1	R2	R3
1	**1**	**3**

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**2**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Tikriname sąlygą **Q(P2, i) <= V(i)** visiems i.

.

	Q(P2)
R1	1
R2	0
R3	1

V
1
1
3

Bankininko algoritmas

N(j,i)

	Dar reikia			Jau yra išskirta		
	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0
P2	1	0	2	5	1	1
P3	1	0	3	2	1	1
P4	4	2	0	0	0	2

A(j,i)

V(i)

Laisvas kiekis		
R1	R2	R3
1	**1**	**3**

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Tikriname sąlygą **Q(P2, i) <= V(i)** visiems i.

.	Q(P2)		V
R1	1	=	1
R2	0	<	1
R3	1	<	3

Sąlyga tenkinama, **laisvų resursų užtenka**. Einame į kitą **žingsnį.**

Priešingu atveju nutraukiam **Q(P2) prašymo** vykdymą.

Bankininko algoritmas

N(j,i) **A(j,i)**

	Dar reikia			Jau yra išskirta		
	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0
P2	1	0	2	5	1	1
P3	1	0	3	2	1	1
P4	4	2	0	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
1	**1**	**3**

Algoritmo žingsniai:

- Pretenduojama į tai, kad resurso poreikiai bus tenkinami ir nustatomas naujas resurso išskyrimo būvis:

$$\mathbf{V(i) = V(i) - Q(P2, i)}$$

visiems i

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Bankininko algoritmas

N(j,i)

	Dar reikia			Jau yra išskirta		
	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0
P2	1	0	2	5	1	1
P3	1	0	3	2	1	1
P4	4	2	0	0	0	2

A(j,i)

V(i)

Laisvas kiekis		
R1	R2	R3
1 - 1	**1 - 0**	**3 - 1**

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Pretenduojama į tai, kad resurso poreikiai bus tenkinami ir nustatomas naujas resurso išskyrimo būvis:

$$V(i) = V(i) - Q(P2, i)$$

visiems i

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	1-1	0-0	2-1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**5+1**	**1+0**	**1+1**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Pretenduojama į tai, kad resurso poreikiai bus tenkinami ir nustatomas naujas resurso išskyrimo būvis:

$$A(P2, i) = A(P2, i) + Q(P2, i)$$

$$N(P2, i) = N(P2, i) - Q(P2, i)$$

visiems i

Bankininko algoritmas

N(j,i)

	Dar reikia			Jau yra išskirta		
	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0
P2	0	0	1	**6**	**1**	**2**
P3	1	0	3	2	1	1
P4	4	2	0	0	0	2

A(j,i)

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

W(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus, t.y ar galime sudaryti tokią procesų seką, kuriai esant galėsim visus procesus įvykdyti

$W(i)=V(i), visiems\ i$

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**6**	**1**	**2**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

W(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus:
- Ieškome proceso, kuriam

$N(j,i) <= W(i), visiems\ i$

Tokiu procesu galima imti P2 procesą

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**6**	**1**	**2**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

W(i)

Laisvas kiekis		
R1	R2	R3
0+6	1+1	2+2

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus:
- Galime P2 procesą užbaigti ir jo resursus skelbti laisvais

$W(i)=W(i)+A(2,i)$, *visiems i*

Tokiu procesu galima imti P2 procesą

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**6**	**1**	**2**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

W(i)

Laisvas kiekis		
R1	R2	R3
6+1	2+0	4+0

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus:
- Ieškome kito proceso, kuriam

N(j,i) <=W(i),visiems i

Tokiu procesu galima imti P1 procesą

Užbaigsime P1 procesą *W(i)=W(i)+A(1,i)*

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**6**	**1**	**2**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

W(i)

Laisvas kiekis		
R1	R2	R3
7+2	2+1	4+1

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus:
- Ieškome kito proceso, kuriam

N(j,i) <=W(i),visiems i

Tokiu procesu galima imti P3 procesą

Užbaigsime P3 procesą *W(i)=W(i)+A(3,i)*

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**6**	**1**	**2**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

W(i)

Laisvas kiekis		
R1	R2	R3
9+0	3+0	5+2

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus:
- Ieškome kito proceso, kuriam

N(j,i) <=W(i),visiems i

Tokiu procesu galima imti P4 procesą

Užbaigsime P4 procesą *W(i)=W(i)+A(4,i)*

Bankininko algoritmas

N(j,i)

	Dar reikia		
	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

A(j,i)

	Jau yra išskirta		
	R1	R2	R3
P1	1	0	0
P2	**6**	**1**	**2**
P3	2	1	1
P4	0	0	2

V(i)

Laisvas kiekis		
R1	R2	R3
0	1	2

Q(j,i)

	Prašymas		
	R1	R2	R3
P1	0	0	0
P2	**1**	**0**	**1**
P3	0	0	0
P4	0	0	0

Algoritmo žingsniai:

- Tikrinsime, ar sistemos būvis bus saugus:
- Kadangi visus procesus pavyko užbaigti taikant seką:
- (P2, P1, P3, P4), tai sistemos būvis yra saugus, resursų priskyrimas P2 procesui yra leistinas

Bankininko algoritmo trūkumai

- *Reikalauja, kad resursų kiekis būtų fiksuotas.*
- *Reikalauja , kad procesų kiekis būtų fiksuotas.*
- *Reikalaujama, kad visos užklausos būtų patenkinamos per baigtinį laiko intervalą.*
- *Reikalauja, kad procesai grąžintų paimtus resursus per baigtinį laiko intervalą.*
- *Reikalauja, kad procesai paskelbtų savo maksimalius poreikius resursams iš anksto.*

Mirties taško nustatymas

- Šiuo atveju procesų užklausos resursams visad yra tenkinamos (jei tik užtenka resursų - tai yra jei tik galima). Operacinei sistemai tokiu atveju reikia:
  - *algoritmo patikrinimui*, ar nėra susidaręs mirties taškas.
  - algoritmo, kuris nusakytų, *kaip išeiti iš šio mirties taško*.
- Tikrinimas, ar yra susidaręs mirties taškas gali būti atliekamas kartu su kiekviena užklausa resursams. Aišku, toks dažnas tikrinimas vartos daug CPU laiko.

Mirties taško nustatymo algoritmas

- Naudosime tas pačias matricas bei vektorius kaip ir resursų priskyrimo algoritme.
- Bus atžymimi visi procesai, kurie nėra įėję į mirties taško situaciją. Pradžioje visi procesai yra nepažymėti.
- Atliekama:
- **Atžymimas** kiekvienas j-tas procesas, kuriam išskirtas resursų kiekis **A(j,i)=0**, kiekvienam **i**-tajam resurso tipui (kadangi šie procesai nėra mirties taške).
- Nustatomas darbinis vektorius **W(i)=V(i)**, visoms **i** reikšmėms.
- Kartojama:
- Randamas nepažymėtas procesas **j**, kuriam **Q(j,i)<=W(i)** visoms **i** reikšmėms. Sustojama, jei tokio **j** proceso nėra.
- Jei toks **j** procesas yra: **j**-tasis procesas pažymimas ir nustatoma **W(i)=W(i)+A(j,i)**, visoms **i**.
- Grįžtama į kartojimą.
- Gale: kiekvienas nepažymėtas procesas yra mirties taške.

Komentarai:

- Procesas **j** nėra mirties taške, jei **Q(j,i)<=W(i)**, visoms **i** reikšmėms. Jei esame optimistai ir priimame, kad **j**-tasis procesas nereikalaus daugiau resursų savo įvykdymui, taigi greitu laiku šis procesas grąžins jam išskirtus resursus, ko pasekoje turėsim:
- **W(i)=W(i)+A(j,i)** visoms **i** reikšmėms.
- Jei ši prielaida nebus teisinga, tai mirties taškas galės įvykti vėliau. Šį mirties tašką galės aptikti mirties taško nustatymo algoritmas, kai jis sekantį kartą bus iškviestas.

Panagrinėkime pavyzdį

- Tarkime, kad turime keturis procesus **P1**, **P2**, **P3** ir **P4** bei penkis resursus: **R1**, **R2**, **R3**, **R4**, **R5**. Esamas resursų paskirstymas pateiktas paveiksle
- Pažymimas procesas **P4**, kadangi jis neturi jam priskirtų resursų.
- Nustatom **W=(0,0,0,0,1)**
- **P3** užklausa **(0,0,0,0,1) <=W**.Taigi pažymim **P3** ir nustatom naują **W** reikšmę:
- **W:=W+(0,0,0,1,0)=(0,0,0,1,1)**
- Kadangi procesų **P1** ir **P2** užklausos yra didesnės nei **W** reikšmė algoritmas baigiasi. **P1** ir **P2** yra mirties taške

	Reikės				
	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Išskirta				
R1	R2	R3	R4	R5
1	0	1	1	0
1	1	0	0	0
0	0	0	1	0
0	0	0	0	0

Esami resursai				
R1	R2	R3	R4	R5
0	0	0	0	1

Išėjimas iš mirties taško

- Aptikus susidariusį mirties tašką *žudomas* procesas, esantis mirties taško situacijoje.
- Panaudojamas procesų *suspendavimo/atnaujinimo* mechanizmas
  - Pavyzdžiui, procesai iškeliami į swap sritį
- Procesų *vykdymas grąžinamas atgal*
  - Reikia išsaugoti tam tikrų kontrolinių taškų kontekstą
  - Neprarandamas su procesu atliktas darbas