



OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

Meziprocesní komunikace a synchronizace procesů

Ing. Pavel Smolka, Ph.D.

Problém uváznutí

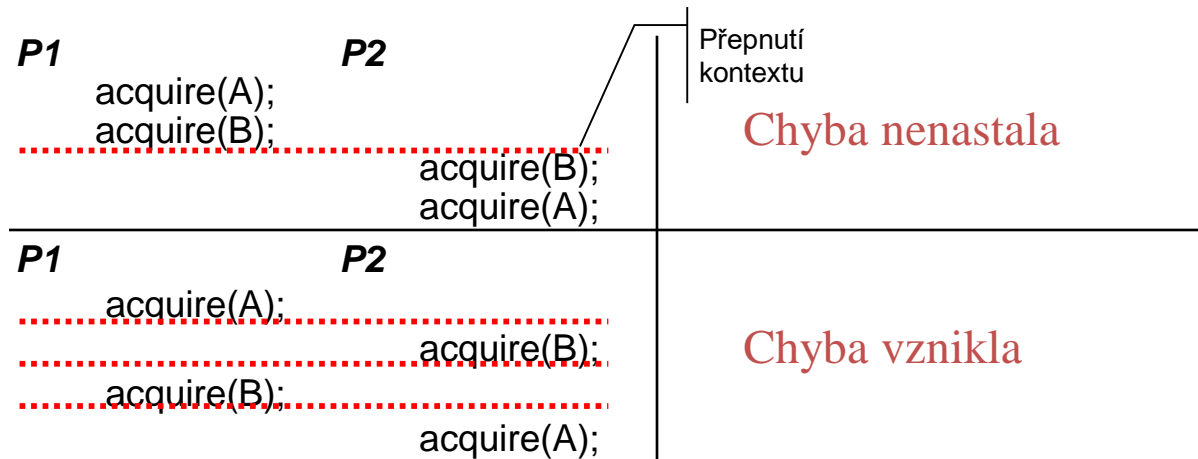
- Existuje množina blokových procesů, z nichž každý vlastní nějaký prostředek (zdroj) a čekajících na zdroj držený jiným procesem z této množiny
- Příklad 1
 - V systému jsou dvě magnetopáskové jednotky
 - V systému existují dva procesy
 - První proces vlastní první mechaniku a potřebuje druhou, druhý proces vlastní druhou mechaniku a potřebuje první
- Příklad 2 – Uváznutí při výměně zpráv
 - Dva mailboxy $bx1$ a $bx2$, operace `receive()` je synchronní (blokuje)

P_0
`receive(bx1, msg);`
...
...
`send(bx2, msg);`

P_1
...
`receive(bx2, msg);`
`send(bx1, msg);`
...

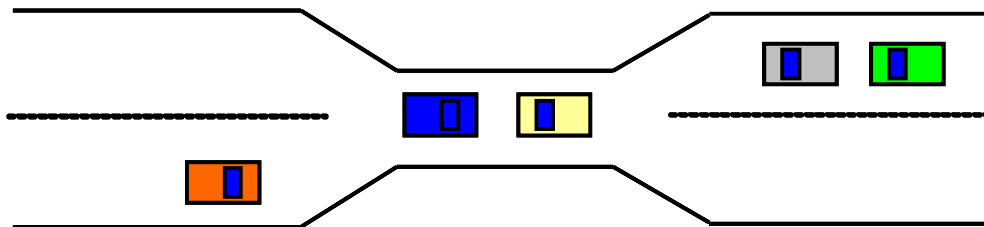
Časově závislé chyby

- Příklad časově závislé chyby
 - Procesy $P1$ a $P2$ spolupracují za použití mutexů A a B



- Nebezpečnost takových chyb je v tom, že vznikají jen zřídka za náhodné souhry okolností
 - Jsou fakticky neodladitelné

Uváznutí na mostě



- Most se střídavým provozem
 - Každý z obou směrů průjezdu po mostě lze chápat jako sdílený prostředek (zdroj)
 - Dojde-li k uváznutí, situaci lze řešit tím, že se jedno auto vrátí – preempce zdroje (přivlastnění si zdroje, který byl vlastněn někým jiným – *preemption*) a vrácení soupeře před žádost o přidělení zdroje (*rollback*)
 - Při řešení uváznutí může dojít k tomu, že bude muset couvat i více aut
 - Riziko stárnutí (hladovění)



Definice uváznutí a stárnutí

- Uváznutí:
 - Množina procesů P uvázla, jestliže každý proces $P_i \in P$ čeká na událost (zaslání zprávy, uvolnění prostředku, ...), kterou může vyvolat pouze proces $P_j \in P$, $j \neq i$.
 - Prostředek: paměťový prostor, V/V zařízení, soubor nebo jeho část, ...
- Stárnutí:
 - Požadavky jednoho nebo více procesů z P nebudou splněny v konečném čase
 - např. z důvodů priorit, opatření proti uváznutí, atd.

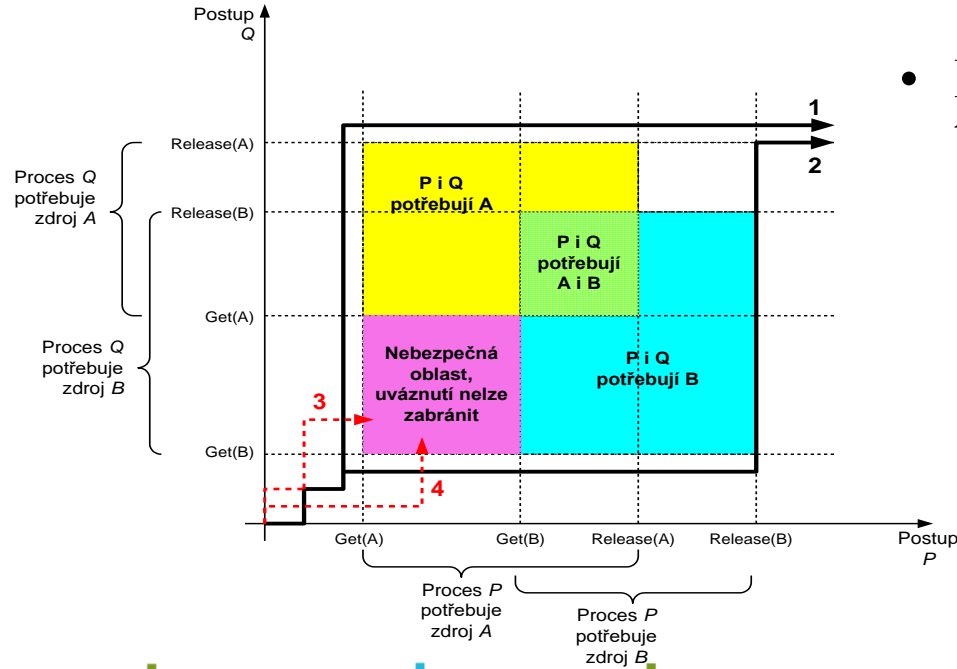


Použitý model systému

- Typy prostředků (zdrojů) $R_1, R_2, \dots R_m$
 - např. časové úseky CPU, úseky v paměti, V/V zařízení,
...
- Každý typ prostředku R_i má W_i instancí
 - např. máme 4 magnetické pásky a 2 CD mechaniky
 - často $W_i = 1$ – tzv. *jednoinstanční prostředky*
- Každý proces používá potřebné zdroje dle následujícího schématu
 - žádost – request, acquire, wait
 - používání prostředku po konečnou dobu (kritická sekce)
 - uvolnění (navrácení) – release, signal

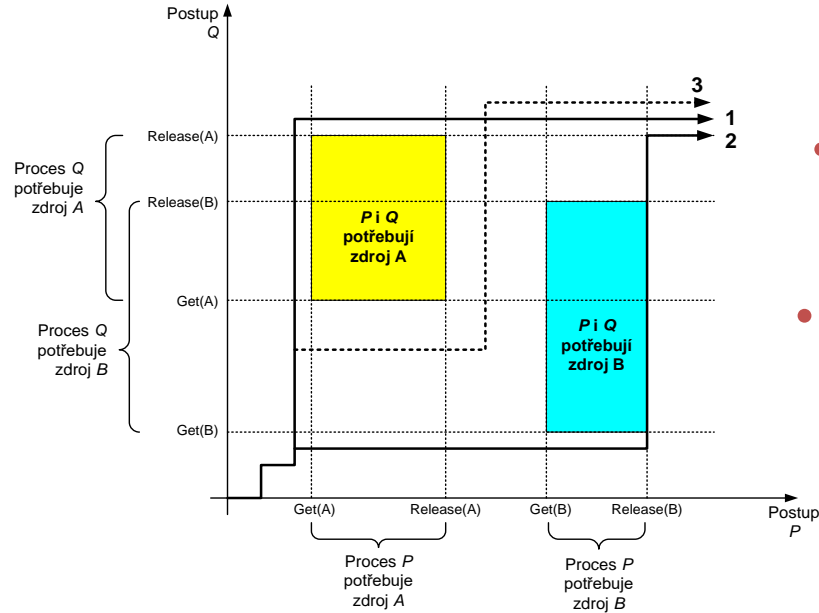


Bezpečné a nebezpečné trajektorie



- Bezpečné a nebezpečné trajektorie procesů
 - 1 procesor – trajektorie vodorovně nebo svisle

Bezpečné trajektorie procesů



- Bezpečné trajektorie
- Z analýzy trajektorií procesů se zdá, že vhodným plánováním procesů lze zabránit uváznutí


Charakteristika uváznutí


- Coffman formuloval čtyři podmínky, které musí platit **současně**, aby uváznutí **mohlo** vzniknout
 1. **Vzájemné vyloučení**, *Mutual Exclusion*
 - sdílený zdroj může v jednom okamžiku používat nejvýše jeden proces
 2. **Postupné uplatňování požadavků**, *Hold and Wait*
 - proces vlastníci alespoň jeden zdroj potřebuje další, ale ten je vlastněn jiným procesem, v důsledku čehož bude čekat na jeho uvolnění
 3. **Nepřipouští se odnímání zdrojů**, *No preemption*
 - zdroj může uvolnit pouze proces, který ho vlastní, a to dobrovolně, když již zdroj nepotřebuje
 4. **Zacyklení požadavků**, *Circular wait*
 - Existuje množina čekajících procesů $\{P_0, P_1, \dots, P_k, P_0\}$ takových, že P_0 čeká na uvolnění zdroje držného P_1 , P_1 čeká na uvolnění zdroje držného P_2, \dots, P_{k-1} čeká na uvolnění zdroje držného P_k , a P_k čeká na uvolnění zdroje držného P_0 .
 - V případě jednoinstančních zdrojů splnění této podmínky značí, že k uváznutí již došlo.



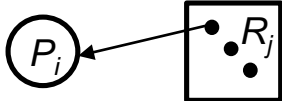
Graf přidělování zdrojů

- Modelování procesů a zdrojů pomocí Grafu přidělování zdrojů (*Resource Allocation Graph*, RAG):
- Množina uzlů V a množina hran E
- Uzly dvou typů:
 - $P = \{P_1, P_2, \dots, P_n\}$, množina procesů existujících v systému
 - $R = \{R_1, R_2, \dots, R_m\}$, množina zdrojů existujících v systému
- Hrany:
 - hrana požadavku – orientovaná hrana $P_i \rightarrow R_j$
 - hrana přidělení – orientovaná hrana $R_j \rightarrow P_i$
- Bipartitní graf

Proces 

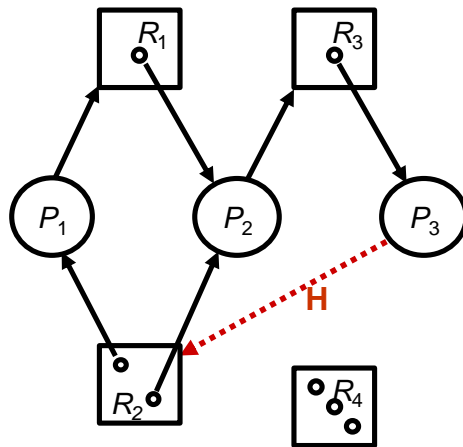
Zdroj typu R_j se 3 instancemi 

Proces P_i požadující prostředek R_j 

Proces P_i vlastníci instanci prostředku R_j 



Příklad RAG



- Proces P_1 vlastní zdroj R_2 a požaduje zdroj R_1
- Proces P_2 vlastní zdroje R_1 a R_2 a ještě požaduje zdroj R_3
- Proces P_3 vlastní zdroj R_3
- Zdroj R_4 není nikým vlastněn ani požadován
- Jednoinstanční zdroje R_1 a R_3 jsou obsazeny
- Instance zdroje R_2 jsou vyčerpány
- Přidání hrany H , kdy proces P_3 zažádá o přidělení zdroje R_2 a zablokuje se, způsobí uvážnutí

- V RAG není cyklus
 - K uvážnutí nedošlo a zatím ani nehrozí
- V RAG se cyklus vyskytuje
 - Jsou-li součástí cyklu pouze zdroje s jednou instancí, pak došlo k uvážnutí
 - Mají-li dotčené zdroje více instancí, pak k uvážnutí může dojít

Plánování procesů a uváznutí

• Uvažme následující příklad:

- 3 procesy soupeří o
- 3 jedno-instanční zdroje

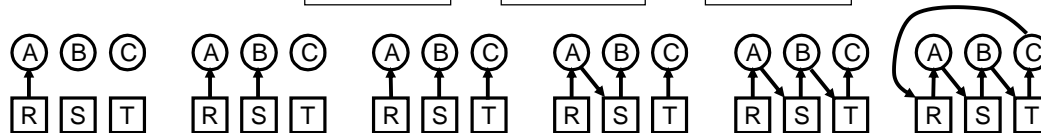
A
Žádá o R
Žádá o S
Uvolňuje R
Uvolňuje S

B
Žádá o S
Žádá o T
Uvolňuje S
Uvolňuje T

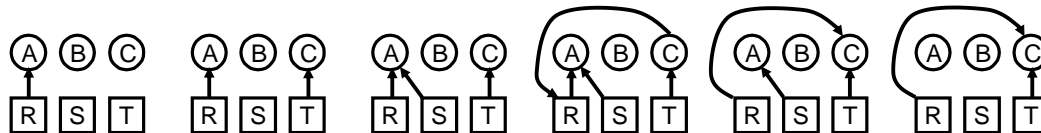
C
Žádá o T
Žádá o R
Uvolňuje T
Uvolňuje R

1. A žádá o R
2. B žádá o S
3. C žádá o T
4. A žádá o S
5. B žádá o T
6. C žádá o R

uváznutí



1. A žádá o R
 2. C žádá o T
 3. A žádá o S
 4. C žádá o R
 5. A uvolňuje R
 6. A uvolňuje S
- uváznutí nenastává
S procesem B již
nejsou problémy



• Lze plánováním předejít uváznutí?

- Za jakých podmínek?
- Jak to algoritmizovat?

Co lze činit s problémem uváznutí?

- Zcela ignorovat hrozbu uváznutí
 - Pštroší algoritmus — strč hlavu do písku a předstírej, že se nic neděje
 - Používá mnoho OS včetně většiny UNIXů
- Prevence uváznutí
 - Pokusit se přijmout taková opatření, aby se uváznutí stalo vysoce nepravděpodobným
- Vyhýbání se uváznutí
 - Zajistit, že k uváznutí *nikdy* nedojde
 - Prostředek se nepřidělí, pokud by hrozilo uváznutí
 - hrozí stárnutí
- Detekce uváznutí a následná obnova
 - Uváznutí se připustí, detekuje se jeho vznik a zajistí se obnova stavu před uváznutím



Prevence před uváznutím

- Konzervativní politikou se omezuje přidělování prostředků
 - Nepřímé metody — narušení některé Coffmanovy podmínky
 - Přímá metoda — plánovat procesy tak, aby nevznikl cyklus v RAG
 - Vzniku cyklu se brání tak, že zdroje jsou očíslovány a procesy je smějí alokovat pouze ve vzrůstajícím pořadí čísel zdrojů
- Nepřímé metody
 - Eliminace potřeby vzájemného vyloučení
 - Nepoužívat sdílené zdroje, virtualizace (spooling) periférií
 - Mnoho činností však sdílení nezbytně potřebuje ke své funkci
 - Eliminace postupného uplatňování požadavků
 - Proces, který požaduje nějaký zdroj, nesmí dosud žádný zdroj vlastnit
 - Všechny prostředky, které bude kdy potřebovat, musí získat naráz
 - Nízké využití zdrojů
 - Připustit násilné odnímání přidělených zdrojů (preempce zdrojů)
 - Procesu žádajícímu o další zdroj je dosud vlastněný prostředek odňat
 - To může být velmi riskantní – zdroj byl již zmodifikován
 - Proces je reaktivován, až když jsou všechny potřebné prostředky volné
 - Metoda inkrementálního zjišťování požadavků na zdroje – nízká průchodnost

Vyhýbání se uváznutí

- Základní problém: Systém musí mít dostatečné apriorní informace o požadavcích procesů na zdroje
 - Nejčastěji se požaduje, aby každý proces udal maxima počtu prostředků každého typu, které bude za svého běhu požadovat
- Algoritmus:
 - Dynamicky se zjišťuje, zda stav subsystému přidělování zdrojů zaručuje, že se procesy v žádném případě nedostanou do cyklu v RAG
- Stav systému přidělování zdrojů je popsán
 - Počtem dostupných a přidělených zdrojů každého typu a
 - Maximem očekávaných žádostí procesů
 - Stav může být bezpečný nebo nebezpečný



Vyhýbání se uváznutí – bezpečný stav

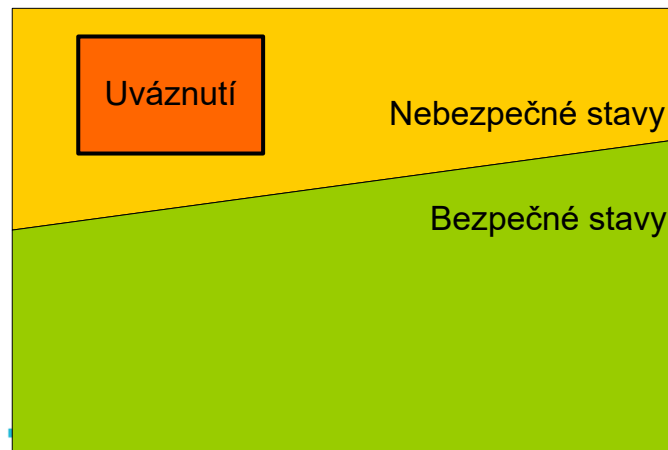
- Má-li být procesu přidělen dostupný prostředek, systém musí rozhodnout, zda toto přidělení zachová systém v „bezpečném stavu“
- Systém je v bezpečném stavu, existuje-li „*bezpečná posloupnost procesů*“
 - Posloupnost procesů $\{P_0, P_1, \dots, P_n\}$ je bezpečná, pokud požadavky každého P_i lze uspokojit právě volnými zdroji a zdroji vlastněnými všemi P_k , $k < i$
 - Pokud nejsou zdroje požadované procesem P_i volné, pak P_i bude čekat dokud se všechny P_k neukončí a nevrátí přidělené zdroje
 - Když P_{i-1} skončí, jeho zdroje může získat P_i , proběhnout a jím vrácené zdroje může získat P_{i+1} , atd.



Vyhýbání se uváznutí – obecné poznatky

17

- Je-li systém v bezpečném stavu (*safe state*) k uváznutí nemůže dojít
- Jestliže je systém ve stavu, který není bezpečný (*unsafe state*), přechod do uváznutí hrozí
- Vyhýbání se uváznutí znamená:
 - Plánovat procesy tak, aby systém byl stále v bezpečném stavu
 - Nespouštět procesy, které by systém z bezpečného stavu mohly vyvést
 - Nedopustit potenciálně nebezpečné přidělení prostředku



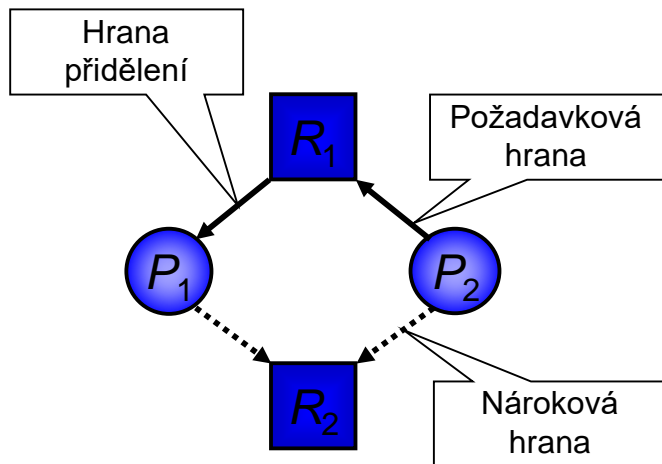
Vyhýbání se uváznutí – algoritmus

- Do RAG se zavede „nároková hrana“
 - Nároková hrana $P_i \rightarrow R_j$ značí, že někdy v budoucnu bude proces P_i požadovat zdroj $P_i \rightarrow R_j$
 - V RAG hrana vede stejným směrem jako požadavek na přidělení, avšak kreslí se čárkovaně
 - Nároková hrana se v okamžiku vzniku žádosti o přidělení převede na požadavkovou hranu
 - Když proces zdroj získá, požadavková hrana se změní na hranu přidělení
 - Když proces zdroj vrátí, hrana přidělení se změní na požadavkovou hranu
 - Zdroje musí být v systému nárokovány předem
 - Převod požadavkové hrany v hranu přidělení nesmí v RAG vytvořit cyklus (včetně uvažování nárokových hran)

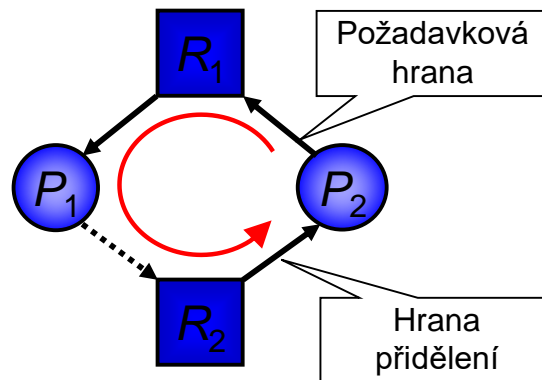


Vyhýbání se uváznutí – algoritmus (2)

19



Tento stav není
bezpečný



Bankéřský algoritmus

- Chování odpovědného bankéře:
 - Klienti žádají o půjčky do určitého limitu
 - Bankér ví, že ne všichni klienti budou svůj limit čerpat současně a že bude půjčovat klientům prostředky postupně
 - Všichni klienti v jistém okamžiku svého limitu dosáhnou, avšak nikoliv současně
 - Po dosažení přislíbeného limitu klient svůj dluh v konečném čase vrátí
 - Příklad:
 - Ačkoliv bankér ví, že všichni klienti budou dohromady potřebovat 22 jednotek, na celou transakci má jen 10 jednotek

Klient	Užito	Max.
Adam	0	6
Eva	0	5
Josef	0	4
Marie	0	7

K dispozici: 10
Počáteční stav (a)

Klient	Užito	Max.
Adam	1	6
Eva	1	5
Josef	2	4
Marie	4	7

K dispozici: 2
Stav (b)

Klient	Užito	Max.
Adam	1	6
Eva	2	5
Josef	2	4
Marie	4	7

K dispozici: 1
Stav (c)

Bankéřský algoritmus (2)

- Procesy
 - Zákazníci přicházející do banky pro úvěr předem deklarují maximální výši, kterou si budou kdy chtít půjčit
 - Úvěry v konečném čase splácí
- Bankéř úvěr neposkytne, pokud si není jist, že uspokojí všechny zákazníky
- Lze použít pro vyhýbání se uváznutí i při zdrojích s více instancemi
- Procesy musí deklarovat své potřeby předem
- Proces požadující přidělení může být zablokován
- Proces všechny přidělené zdroje vrátí v konečném čase
- Nikdy nedojde k uváznutí
 - Proces bude spuštěn jen, pokud bude možno uspokojit všechny jeho požadavky
- Sub-optimální pesimistická strategie
 - Předpokládá se nejhorší případ
 - Procesy musí zadat své požadavky předem a všechny naráz



Bankéřský algoritmus – datové struktury

- n ... počet procesů
 - m ... počet typů zdrojů
 - Vektor `available[m]`
 - `available[j] = k` značí, že je k instancí zdroje typu R_j je volných
 - Matice `max[n, m]`
 - Povinná deklarace procesů:
 - `max[i, j] = k` znamená, že proces P_i bude během své činnosti požadovat až k instancí zdroje typu R_j
 - Matice `allocated[n, m]`
 - `allocated[i, j] = k` značí, že v daném okamžiku má proces P_i přiděleno k instancí zdroje typu R_j
 - Matice `needed[n, m]`
 - `needed[i, j] = k` říká, že v daném okamžiku procesu P_i chybí ještě k instancí zdroje typu R_j
- Platí $\text{needed}[i, j] = \text{max}[i, j] - \text{allocated}[i, j]$



Procedura otestování bezpečnosti stavu

1. Necht'

- $work[m]$ a $finish[n]$ jsou pracovní vektory
- Inicializujeme $work = available$; $finish[i] = false$; $i=1, \dots, n$

2. Najdi i , pro které platí

- $(finish[i] == false) \ \&\& \ (needed[i] \leq work[i])$
- Pokud takové i neexistuje, jdi na krok 4

3. Simuluj ukončení procesu i

- $work[i] += allocated[i]$; $finish[i] = true$;
- Pokračuj krokem 2

4. Pokud platí

- $finish[i] == true$ pro všechna i , pak stav systému je bezpečný



Postup při žádosti o přidělení zdroje

Nechť request je požadavkový vektor procesu P_i :

request[j] == k znamená, že proces P_i žádá o k instancí zdroje typu R_j

1. if(request[j] >= needed[i, j]) error;
 - Deklarované maximum překročeno!
2. if(request[j] <= available[j]) goto 3;
 - Jinak zablokuj proces P_i – požadované prostředky nejsou volné
3. Namodeluj přidělení prostředku a otestuj bezpečnost stavu:
 - available[j] = available[j] – request[j];
 - allocated[i, j] = allocated[i, j] + request[j];
 - needed[i, j] = needed[i, j] – request[j];

}

Akce 3

 - Spuště test bezpečnosti stavu
 - Je-li bezpečný, přiděl požadované zdroje
 - Není-li stav bezpečný, pak vrať úpravy „Akce 3“ a zablokuj proces P_i , neboť přidělení prostředků by způsobilo nebezpečí uvážnutí

Bankéřský algoritmus – příklad

- 5 procesů P_0 až P_4 ,
- Zdroje tří typů: A v 10 instancích, B – 5 instancí a C má 7 instancí
- Snímek v čase T_0 :

	allocated				max				needed				available		
	A	B	C		A	B	C		A	B	C		A	B	C
P_0	0	1	0		7	5	3		7	4	3		3	3	2
P_1	2	0	0		3	2	2		1	2	2				
P_2	3	0	2		9	0	2		6	0	0				
P_3	2	1	1		2	2	2		0	1	1				
P_4	0	0	2		4	3	3		4	3	1				

- Stav je bezpečný
 - Existuje posloupnost (např. $\langle P_1, P_3, P_4, P_0, P_2 \rangle$), která je bezpečná

Bankéřský algoritmus – příklad – pokr.

- Proces P_1 žádá o zdroje (1, 0, 2)
- Kontrola přípustnosti:
 - request \leq available, tj. (1, 0, 2) $<$ (3, 3, 2) – splněno
- Simulace přidělení

	allocated				max				needed				available		
	A	B	C		A	B	C		A	B	C		A	B	C
P_0	0	1	0		7	5	3		7	4	3		2	3	0
P_1	3	0	2		3	2	2		0	2	0				
P_2	3	0	2		9	0	2		6	0	0				
P_3	2	1	1		2	2	2		0	1	1				
P_4	0	0	2		4	3	3		4	3	1				

- Stav je bezpečný
 - Existuje posloupnost (např. $\langle P_1, P_3, P_4, P_2, P_0 \rangle$), která je bezpečná
- Bylo možno přidělit zdroje (3, 3, 0) procesu P_4 ?

Pojmy

- Stav systému je definován hodnotami
 - $R(i)$ co existuje
 - $C(j,i)$ co se požaduje pro typy prostředků i a procesy j
- Počet přidělených prostředků typu i procesu j
 - $A(j,i)$ pro všechna (j,i)
- Souhrnný počet dostupných prostředků typu i
 - $V(i)=R(i)-\text{SUMA } A(k,i)$
- Počet prostředků typu i potřebných pro dokončení procesu j
 - $N(j,i)=C(j,i)-A(j,i)$
- Bankéř prostředek přidělí, pokud systém po přidělení zůstane v bezpečném stavu
- Požadovaný počet prostředků typu i procesem j
 - $Q(j,i)$

Rámcový algoritmus

IF $Q(j,i) \leq N(j,i)$ pro všechna i THEN pokračuj
ELSE oznámení chyby procesu /*nesplnitelný požadavek*/

IF $Q(j,i) \leq V(j,i)$ pro všechna i THEN pokračuj
ELSE čekej /*prostředek i není dostupný, čekej na jeho uvolnění*/

$V(j,i) = V(i) - Q(j,i)$ pro všechna i

$A(j,i) = A(j,i) + Q(j,i)$ pro všechna i

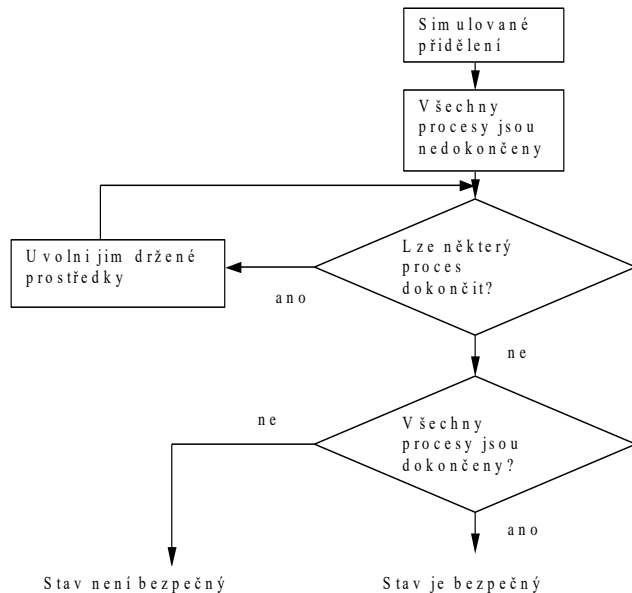
$N(j,i) = N(j,i) - Q(j,i)$ pro všechna i

IF výsledný stav je bezpečný, THEN prostředek se procesu j přidělí

ELSE proces j musí čekat na splnění požadavku $Q(j,i)$ a obnoví se původní stav



Algoritmus bankéře



Detailní algoritmus

INICIALIZACE: všechny procesy se prohlásí za neukončené;
nastaví se pracovní vektor počtu dostupných prostředků $W(i)$;
 $W(i)=V(i)$ pro všechna i ;
REPEAT: najezni neukončený proces j , který má počet prostředků typu i potřebných
pro dokončení $N(j,i) \leq W(i)$ pro všechna i
 $N(j,i)=C(j,i)-A(j,i)$ počet prostředků typu i potřebných pro dokončení
procesu j
IF takový proces j neexistuje
THEN GO TO EXIT
ELSE označ takový proces za ukončený a uvolni jeho prostředky:
 $W(i)=W(i)+A(j,i)$ pro všechna i
GO TO REPEAT
EXIT: stav:=IF všechny procesy ukončené THEN BEZPEČNÝ
ELSE NENÍ BEZPEČNÝ



Příklad použití

- Prostředků m
- Procesů n
- Algoritmus určuje, zda paralelní stav je jistý před zablokováním, prověřováním m požadavků od každého procesu, dokud není nalezen takový proces, který může být dokončen.
- Algoritmus opakuje tuto prověrku, dokud není eliminováno všech n procesů
- Maximálně je doba úměrná

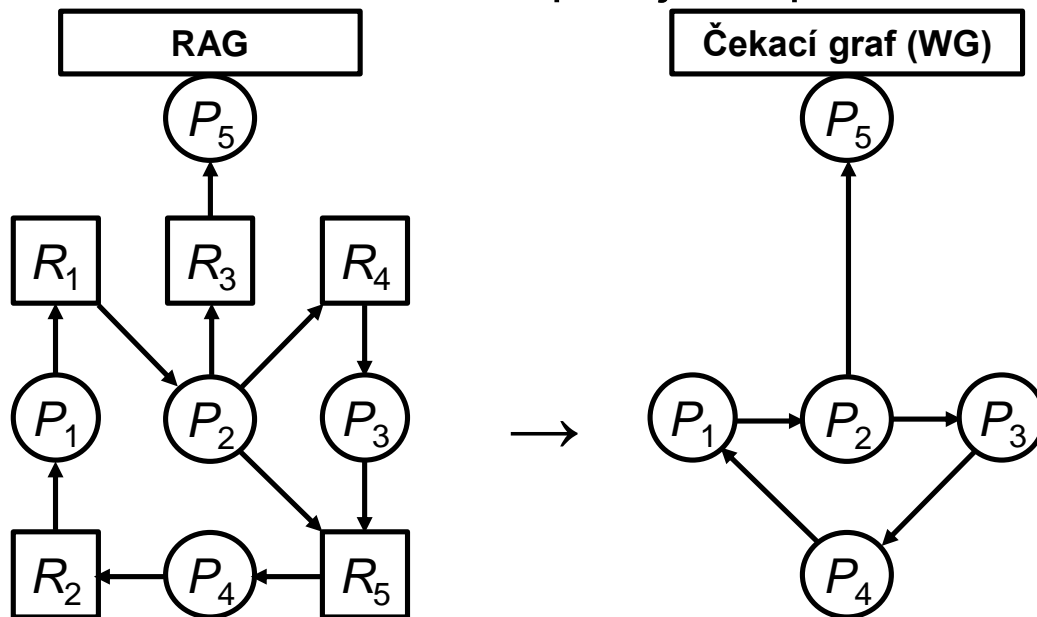
$$m(n+n-1+n-2+\dots+1)=mn(n+1)/2$$



Detekce uváznutí s následnou obnovou

32

- Strategie připouští vznik uváznutí:
 - Uváznutí je třeba detekovat
 - Vznikne-li uváznutí, aplikuje se plán obnovy systému



Detekce uváznutí – postup

- Příklad jednoinstančního zdroje daného typu
 - Udržuje se čekací graf – uzly jsou procesy
 - Periodicky se provádí algoritmus hledající cykly
 - Algoritmus pro detekci cyklu v grafu má složitost $O(n^2)$, kde n je počet hran v grafu
- Příklad více instancí zdrojů daného typu
 - n ... počet procesů
 - m ... počet typů zdrojů
 - Vektor `available[m]`
 - `available[j] = k` značí, že je k instancí zdroje typu R_j je volných
 - Matice `allocated[n, m]`
 - `allocated[i, j] = k` značí, že v daném okamžiku má proces P_i přiděleno k instancí zdroje typu R_j
 - Matice `request[n, m]`
 - Indikuje okamžité požadavky každého procesu:
 - `request[i, j] = k` znamená, že proces P_i požaduje dalších k instancí zdroje typu R_j



Detekce uváznutí – algoritmus

1. Necht'
 - $work[m]$ a $finish[n]$ jsou pracovní vektory
 - Inicializujeme $work = available$; $finish[i] = false$; $i=1, \dots, n$
2. Najdi i , pro které platí
 - $(finish[i] == false) \ \&\& \ (request[i] \leq work[i])$
 - Pokud takové i neexistuje, jdi na krok 4
3. Simuluj ukončení procesu i
 - $work[i] += allocated[i]$; $finish[i] = true$;
 - Pokračuj krokem 2
4. Pokud platí
 - $finish[i] == false$ pro některé i , pak v systému došlo k uváznutí. Součástí cyklů ve WG jsou procesy P_i , kde $finish[i] == false$

Algoritmus má složitost $O(m n^2)$

- Výpočetně značně náročné



Detekce uváznutí – příklad

- 5 procesů P_0 až P_4 ,
- Zdroje tří typů: A (7 instancí), B (2 instance) a C (6 instancí)
- Snímek v čase T_0 :

	allocated				request				available		
	A	B	C		A	B	C		A	B	C
P_0	0	1	0		0	0	0		0	0	0
P_1	2	0	0		2	0	2				
P_2	3	0	3		0	0	0				
P_3	2	1	1		1	0	0				
P_4	0	0	2		0	0	2				

- Existuje posloupnost $\langle P_0, P_2, P_3, P_1, P_4 \rangle$, která končí
s $\text{finish}[i] == \text{true}$ pro všechna i

Detekce uváznutí – příklad

- Nechť nyní P_2 požaduje další exemplář zdroje typu C

	request		
	A	B	C
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- Jaký je stav systému?
 - P_2 sice může získat zdroj typu C od procesu P_1 , avšak požadavky ostatních procesů nelze uspokojit
 - System uváznul, uváznutí se týká procesů P_1 , P_2 , P_3 a P_4

Použitelnost detekčního algoritmu

- Kdy a jak často algoritmus vyvolávat?
 - Je výpočetně náročný
 - Jak často bude uváznutí vznikat?
 - Kolika procesů se uváznutí týká a kolik jich bude muset být „likvidováno“?
 - Minimálně jeden v každém disjunktním cyklu ve WG
- Bude-li se algoritmus volat náhodně, pak
 - cyklů může být velmi mnoho a nebudeme schopni určit proces, který je viníkem uváznutí tolika procesů



Obnova po uváznutí

- Násilné ukončení všech uváznutých procesů
 - velmi tvrdé a nákladné
- Násilně se ukončují dotčené procesy dokud cyklus nezmizí
 - Jak volit pořadí ukončování
 - Kolik procesů bude nutno ukončit
 - Jak dlouho už proces běžel a kolik mu zbývá do ukončení
 - Je to proces interaktivní nebo dávkový (dávku lze snáze restartovat)
 - Cena zdrojů, které proces použil
 - Výběr oběti podle minimalizace ceny
 - Nebezpečí stárnutí
 - některý proces bude stále vybírán jako oběť



Závěrečné úvahy o uváznutí

- Metody popsané jako „prevence uváznutí“ jsou velmi restriktivní
 - ne vzájemnému vyloučení, ne postupnému uplatňování požadavků, preempce prostředků
- Metody „vyhýbání se uváznutí“ nemají dost apriorních informací
 - zdroje dynamicky vznikají a zanikají (např. úseky souborů)
- Detekce uváznutí a následná obnova
 - jsou vesměs velmi drahé – vyžadují restartování aplikací
- Smutný závěr
 - **Problém uváznutí je v obecném případě efektivně neřešitelný**
- Existuje však řada algoritmů pro speciální situace



Speciální algoritmy

- Postup na aplikační úrovni
 - Vyžaduje modifikovanou operaci `acquire` — `try_acquire`, která při momentální nedostupnosti zdroje proces nezablokuje, ale vrátí řízení volajícímu procesu s indikací, že zdroj je obsazen
 - Zjistí-li proces, že zdroj je nedostupný, pak vrátí (`release`) všechny vlastněné prostředky a chvíli počká. Poté zkouší získat potřebné zdroje znovu
 - Výhody
 - jednoduché
 - Nevýhody
 - nízká průchodnost
 - programátorská disciplína
 - použitelné beze ztrát jen pokud proces některý z dosud vlastněných zdrojů nezmodifikoval



Speciální algoritmy (2)

- Postupy používané v databázových systémech
 - Transakční mechanismy
 - Transakce je sada operací, které musí být provedeny jako jediná logická akce (analogie s kritickými sekcemi)
 - Transakce je v databázových systémech série logicky spolu svázaných operací read a write
 - Úspěšně provedená transakce končí operací commit
 - Pokud transakci nelze dokončit (např. pro konflikt přístupu k datům), končí se operací abort
 - Neúspěšná transakce musí „odčinit“ již provedené modifikace (roll-back) nebo jinak vrátit data do původního stavu
 - Lze aplikovat pouze na znovu-použitelné prostředky
 - Kontra příklad: Potištěný papír již nikdy nebude čistý
 - Detaily – viz specializované předměty o databázích



Děkuji za pozornost

