Operační systémy

Uváznutí vláken.

Jan Trdlička

České vysoké učení technické v Praze, Fakulta informačních technologií
Katedra počítačových systémů

https://courses.fit.cvut.cz/BI-OSY

Obsah přednášky

1. Výpočetní prostředky
   - Alokace/uvolnění
   - Alokační graf
2. Uváznutí
   - Definice
   - Coffmanovy podmínky
3. Strategie pro řešení uváznutí
   - Pštrosí strategie
   - Prevence uváznutí
   - Předcházení vzniku uváznutí
   - Detekce a zotavení

Výpočetní prostředky

- **Výpočetní prostředek**
  - fyzický (např. fyzická paměť, tiskárna,. . .),
  - logický (např. proměnná, soubor, mutex,. . .).
- **Sdílené výpočetní prostředky**
  - Pokud je prostředek sdílen (používán) více vlákny současně, mohou vznikat časově závislé chyby $\Rightarrow$ je nutné zajistit výlučný přístup k prostředku (v jednom okamžiku může být používán pouze jedním vláknem).
- **Paralelní přístup k více sdíleným prostředkům**
  - Vlákna velmi často potřebují přistupovat k více prostředkům současně.
- **Typy sdílených prostředků**
  - Odnímatelné (preemptable): již alokovaný prostředek může být vláknu odebrán bez rizika dalších problémů (např. odložení procesu z fyzické paměti na disk při nedostatku fyzické paměti).
  - Neodnímatelné (nonpreemptable): nemohou být odebrány bez rizika (např. tiskárna,...).
  - Bohužel většina prostředků je neodnímatelná.

Výpočetní prostředky

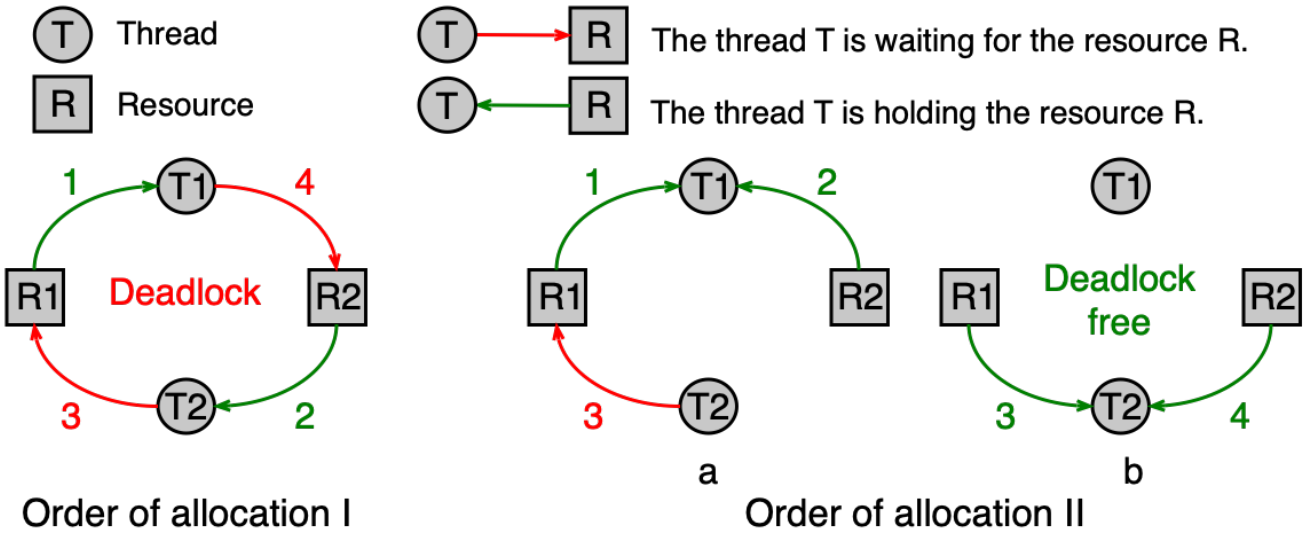
- **Sekvence kroků při použití sdíleného prostředku vláknem**
  1. alokace,
  2. použití,
  3. uvolnění.
- **Alokace prostředku**
  - Vlákno žádá a prostředek prostřednictvím alokační funkce.
  - Pokud je prostředek volný, pak je přidělen danému vláknu.
  - Pokud je prostředek již alokovaný, pak existuje několik scénářů:
    a. Vlákno bude blokováno, dokud prostředek nebude k dispozici (např. `mutex_lock()`, `cond_wait()`, `sam_wait()`,...).
    b. Vlákno bude blokováno maximálně po určitý čas (např. `mutex_timedlock()`,...).
    c. Vlákno nebude blokováno (např. `fork()`, `malloc()`,...).
  - V případech b a c vlákno zjistí např. na základě návratové hodnoty funkce, zda mu byl prostředek přidělen, či nikoliv. Vlákno potom musí rozhodnout, jak bude výpočet dál pokračovat.

Výpočetní prostředky

- **Uvolnění prostředku**
  - Vlákna by sama měla uvolňovat alokované prostředky!
  - Některé prostředky uvolňuje jádro OS automaticky v okamžiku zániku procesu (např. paměť alokovaná pomocí `malloc()`).
  - Jiné prostředky se neuvolní ani po zániku procesu (např. sdílená paměť alokovaná pomocí `shmget()`).
- V tomto textu budeme vycházet z následujících předpokladů.
  - Pokud nejsou prostředky volné, potom vlákna budou čekat dokud se požadované prostředky neuvolní.
  - Alokované prostředky budou používány vlákny po konečnou dobu a potom je vlákna sama uvolnění.
- **Uváznutí (deadlock)**
  - Situace, kdy několik vláken čeká na událost/prostředek, kterou může vyvolat/uvolnit pouze jedno z čekajících vláken.

Alokační graf

- Znázorňuje alokaci prostředků jednotlivými vlákny.
- Je to orientovaný graf s dvěma typy uzlů (prostředky/vlákna).
- Hrany grafu mají orientaci
  - od prostředku k vláknu, pokud vlákno má prostředek již alokovaný,
  - od vlákna k prostředku, pokud vlákno čeká na daný prostředek.
- Číslo hrany může reprezentovat pořadí alokace.



- Každá smyčka v grafu představuje uváznutí (vlákna ve smyčce čekají a nemohou pokračovat).
- Uváznutí závisí na pořadí v jakém jsou prostředky alokovány.

Coffmanovy podmínky

- Uváznutí nastane pouze pokud jsou **splněny následující podmínky.**
  1. Vzájemné vyloučení: každý prostředek je buď přidělen právě jednomu vláknu a nebo je volný (prostředek nemůže být sdílen více vlákny).
  2. Podmínka neodnímatelnosti: prostředek, který byl již přidělen nějakému vláknu, nemůže mu být násilím odebrán (musí být dobrovolně uvolněn daným vláknem).
  3. Podmínka "drž a čekej": vlákno, které má již přiděleny nějaké prostředky, může žádat o další prostředky (vlákno může žádat o prostředky postupně).
  4. Podmínka kruhového čekání: musí existovat smyčka dvou nebo více vláken, ve které každé vlákno čeká na prostředek přidělený dalšímu vláknu ve smyčce.
- První tři podmínky jsou nutné ale ne dostačující $\Rightarrow$ k uváznutí může dojít. Poslední podmínka představuje samotné uváznutí.
- Pokud aspoň jedna z podmínek není splněna, nemůže dojít k uváznutí.

Strategie pro řešení uváznutí

1. Pštrosí strategie
   - Ignorování celého problému.
2. Prevence uváznutí
   - Pomocí nesplnění aspoň jedné z Coffmanových podmínek.
3. Předcházení vzniku uváznutí
   - Na základě pečlivé alokace prostředků.
4. Detekce uváznutí a zotavení
   - K uváznutí může dojít, ale je detekováno a odstraněno.

Pštrosí strategie

- Strategie, ve které se problém uváznutí neřeší/řeší částečně.
- Pokud k uváznutí dojde je vyžadován zásah uživatele/administrátora.
- **Tato strategie má opodstatnění za následujících podmínek**
  - systém obsahuje velký počet různě se chovajících vláken a velký počet různých prostředků,
  - pravděpodobnost výskytu uváznutí je relativně malá,
    $\Rightarrow$ řešení uváznutí by bylo příliš drahé.
- Praktické řešení pro většinu univerzálních OS (MS Windows, OS unixového typu,...).
  - Jádro OS je navrženo jako "deadlock free".
  - Na úrovni procesů/vláken se problém řeší pouze částečně.
    - ★ Prostředky systému jsou omezené (fyzická paměť, systémy souborů, maximální počet vláken,...).
    - ★ Uživatelské vlákna se chovají nepredikovatelně.
    - ★ Částečně lze řešit různými limity (viz. unixový příkaz `ulimit -a`).
- **Toto řešení není přijatelné v "fault tolerant" systémech.**

Prevence uváznutí

- Tato strategie je založená na porušení aspoň jedné z Coffmanových podmínek $\Rightarrow$ zamezíme vzniku uváznutí.

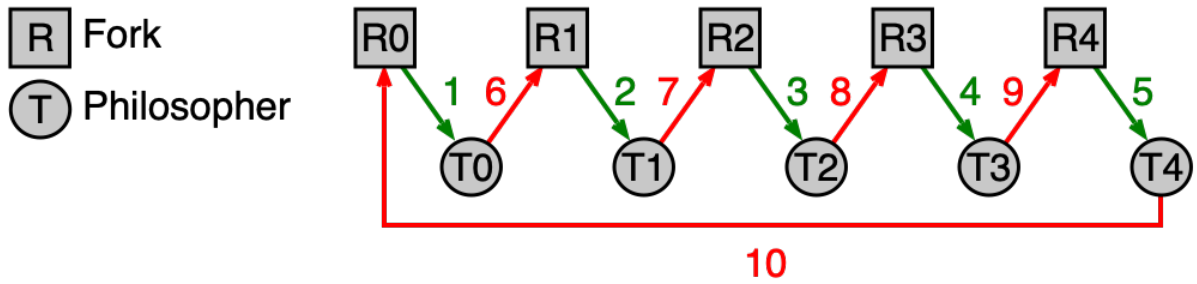
1. **Porušení podmínky "vzájemného vyloučení"**
   - Pokud je prostředek používán více vlákny pro čtení i zápis, pak v praxi tuto podmínku nelze porušit bez rizika vzniku časově závislých chyb.
2. **Porušení podmínky "neodnímatelnosti prostředku"**
   - Tento přístup je vhodný pouze v případech, kdy je možné uložit (zapamatovat) si stav prostředku tak, aby později mohl být opět obnoven do původního stavu.
   - Jako příklad může sloužit způsob sdílení jádra CPU více vlákny (přepínání kontextu).
3. **Porušení podmínky "drž a čekej"**
   - Pokud má vlákno dopředu informaci o tom, které prostředky bude během své existence používat $\Rightarrow$ lze všechny prostředky alokovat najednou v jednom kroku před jejich použitím (vlákno získá vše nebo začne čekat).
   - Tento typ alokace většinou povede k horšímu využití prostředků.

Prevence uváznutí

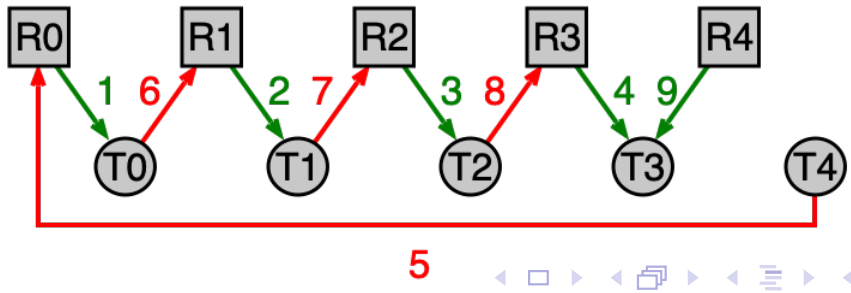
4. **Porušení podmínky "kruhového čekání"**
   - Myšlenka porušení této podmínky je založená na vhodném číslování prostředků a jejich přidělování ve vzestupné pořadí.
   - Každému prostředku přiřadíme jedinečné číslo v rámci systému
     - množina prostředků $R = R_1, \ldots, R_m$,
     - číslování (mapování jedna k jedné) $F : R \rightarrow N$, kde N je množina celých čísel.
   - Vlákno potom může žádat o libovolné prostředky, ale pouze v rostoucím číselném pořadí
     - pokud vlákno žádá o prostředek R_j, pak pro každý prostředek již přidělený vláknu R_i musí platit $F(R_j) > F(R_i)$.
   - Tímto způsobem přidělování nevznikne v alokačním grafu smyčka.
   - Vhodné očíslování prostředků nemusí vždy existovat.
   - Pomocí tohoto přístupu řešíme problém uváznutí ve fázi návrhu/kompilace programu.

Příklad: Prevence "kruhového čekání"

- Uvažujme "naivní řešení" večeřících filosofů
  - Prostředky (vidličky) se alokují pomocí funkce `take_fork(i)`.
  - Pokud vidlička není volná, tak tato funkce zablokuje filosofa, ze kterého byla zavolána.
  - Pokud všichni filosofové "současně" alokovali svou levou vidličku $\Rightarrow$ **řešení s uváznutím**.



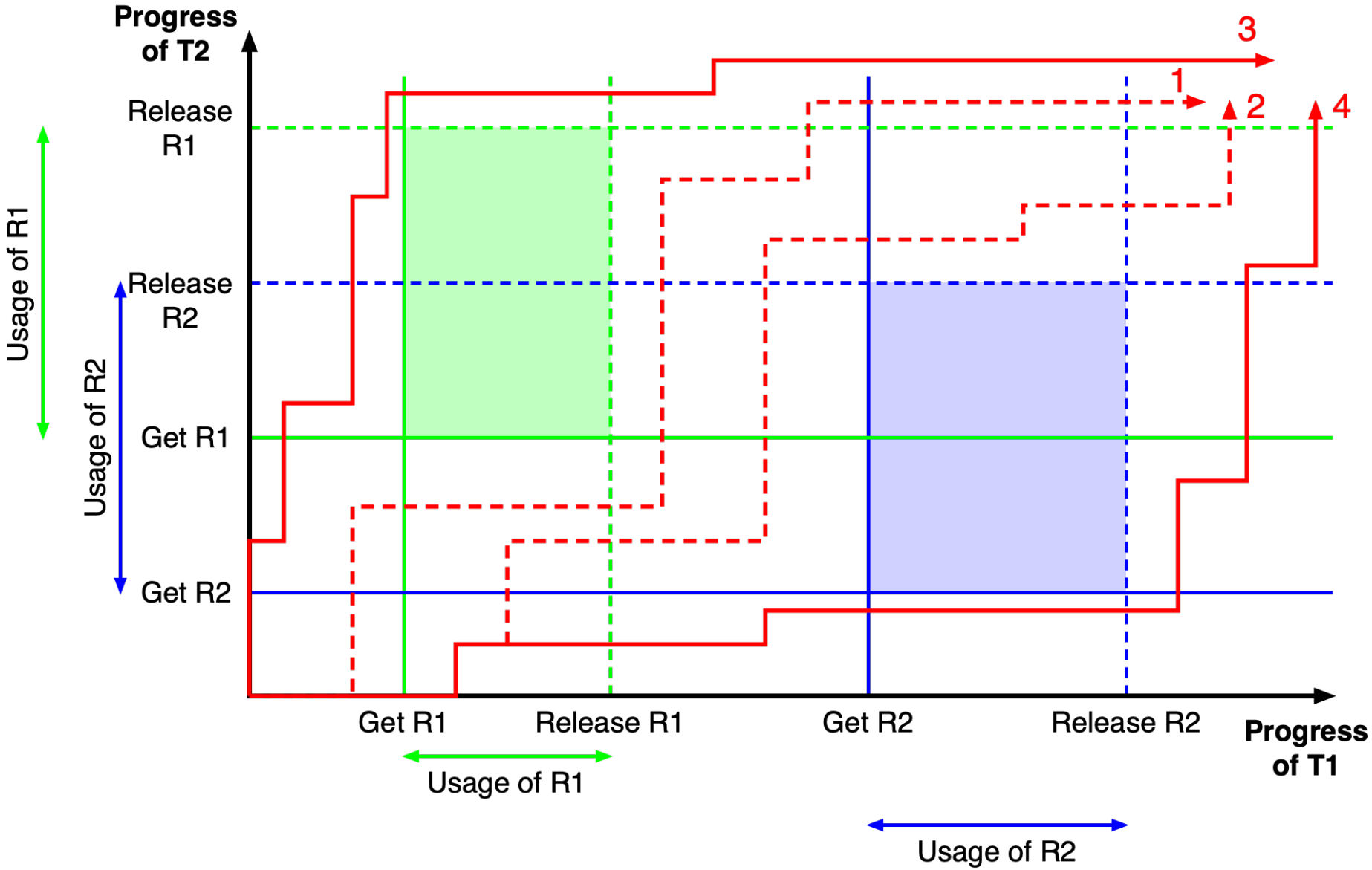
  - Pokud však filosof musí alokovat vidličky ve vzrůstajícím pořadí $\Rightarrow$ **řešení bez uváznutím**.



Příklad 1: Jak předejít vzniku uváznutí

- **Předpokládejme, že máme dvě vlákna T1 a T2.**
  - Vlákno T1 si alokuje dva prostředky v pořadí R1, R2.
  - Vlákno T2 si alokuje stejné prostředky v opačném pořadí R2, R1.
- **Průběh používaní prostředků můžeme znázornit 2D grafem** (viz. následující grafy).
  - Vodorovná osa reprezentuje operace prováděné vláknem T1.
  - Svislá osa reprezentuje operace prováděné vláknem T2.
- Protože každý prostředek může být v jednom okamžiku používán pouze jedním vláknem $\Rightarrow$ vlákna musí obejít zelenou a modrou oblast.
- **Trajektorie 1 až 4**
  - představují postupnou alokaci prostředků bez uváznutí.

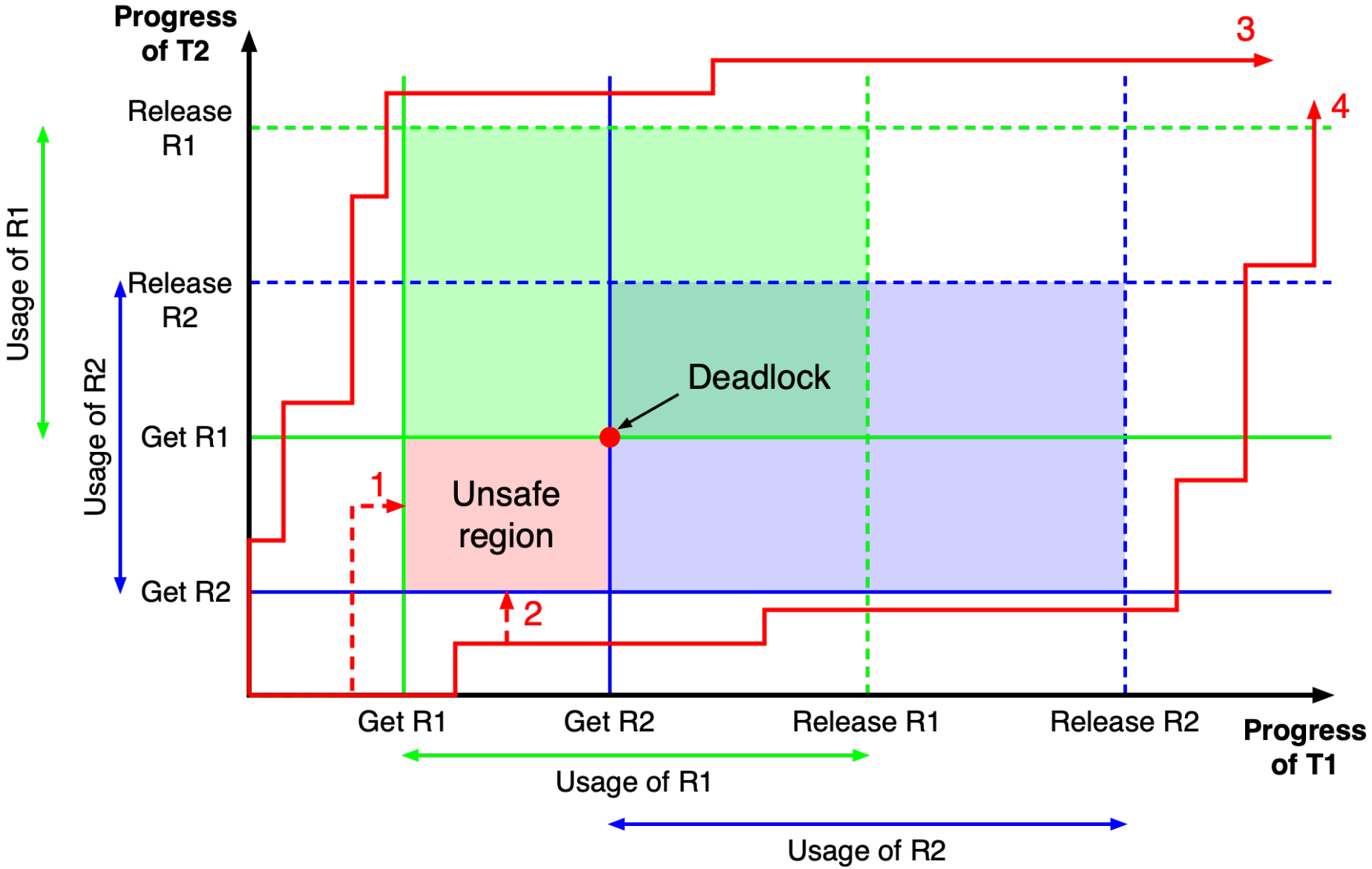
Příklad 1: Jak předejít vzniku uváznutí



Příklad 2: Jak předejít vzniku uváznutí

- **Pokud se změní doba, po kterou vlákno používá prostředek $\Rightarrow$ může dojít uváznutí.**
- **Trajektorie 1 a 2**
  - Červená oblast (unsafe region) představuje pro vlákna past $\Rightarrow$ pokud do ní vstoupí po určitém čase skončí uváznutím.
- **Trajektorie 3 a 4**
  - Pokud červenou, zelenou a modrou oblast vlákna obejdou (vhodnou alokací prostředků) $\Rightarrow$ uváznutí se vyhnou.

Příklad 2: Jak předejít vzniku uváznutí



Předcházení vzniku uváznutí

- **V příkladu 1 nenastal problém s uváznutím.**
  - Důvodem bylo to, že vlákno T1 uvolnilo prostředek R1 dříve než požádalo o prostředek R2 $\Rightarrow$ nevznikla nebezpečná oblast (unsafe region).
  - Důležité je vědět, které prostředky budou vlákna používat a které z nich současně.
- **Jaké znalosti o požadavcích vláken potřebujeme znát?**
  - Předem neznáme požadavky vláken na prostředky
    $\Rightarrow$ uváznutí nelze předejít.
  - Předem známe
    - ⋆ požadavky vláken na prostředky
      $\Rightarrow$ uváznutí lze předejít, ale prostředky nebudou efektivně využity (v příkladu 1 by nebyly povoleny trajektorie 1 a 2).
    - ⋆ požadavky vláken na prostředky + společné používání prostředků
      $\Rightarrow$ uváznutí lze předejít a využití prostředků bude lepší (v příkladu 1 by byly povoleny všechny trajektorie 1 až 4).

Předcházení vzniku uváznutí

- **Popis aktuálního rozdělení prostředků v systému**
  - V systému je n vláken a m různých typů prostředků.
  - **Vektor existujících prostředků** E (existence vector)
  - **Vektor volných prostředků** F (free vector)
  - **Matice požadavků** Q (request matrix)
    - ★ Obsahuje informaci o celkových požadavcích vláken na prostředky.
  - **Matice přidělených prostředků** A (allocation matrix)
    - ★ Obsahuje informaci o aktuálně alokovaných prostředcích.
  - **Matice chybějících prostředků** M (missing matrix)
    - ★ Obsahuje informaci o prostředcích, které vláknům aktuálně chybí.

$$E = [E_1, \ldots, E_m] \qquad F = [F_1, \ldots, F_m]$$

$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \ldots & Q_{1m} \\ Q_{21} & Q_{22} & \ldots & Q_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{n1} & Q_{n2} & \ldots & Q_{nm} \end{bmatrix} \qquad A = \begin{bmatrix} A_{11} & A_{12} & \ldots & A_{1m} \\ A_{21} & A_{22} & \ldots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \ldots & A_{nm} \end{bmatrix}$$

$$M = Q - A$$

Předcházení vzniku uváznutí

- **Pro rozložení prostředků v systému by mělo platit**
  - Všechny prostředky jsou buď volné nebo přidělené vláknům

    $$E_i = F_i + \sum_{k=1}^{n} A_{ki} \quad \text{pro všechna } i = 1, \ldots, m.$$

  - Žádné vlákno nepožaduje více prostředků než kolik jich je v systému

    $$Q_{ki} \leq E_i \text{ pro všechna } k = 1, \ldots, n \text{ a } i = 1, \ldots, m.$$

  - Žádné vlákno si nealokuje více prostředků než požadovalo

    $$A_{ki} \leq Q_{ki} \quad \text{pro všechna } k = 1, \ldots, n \text{ a } i = 1, \ldots, m.$$

Předcházení vzniku uváznutí

- **Předpoklady**
  - Předem známe všechny požadavky vláken na prostředky, které budou během své existence potřebovat.
  - Pokud jim tyto prostředky přidělíme $\Rightarrow$ vlákna tyto prostředky po určité době uvolní.
- **Bezpečný stav**
  - Stav, ve kterém existuje taková posloupnost přidělování prostředků vláknům, která garantuje postupné uspokojení potřeb všech vláken.
- **Bankéřův algoritmus**
  - Když vlákno požádá o prostředek
    - ★ prostředek bude přidělen pokud systém zůstane v bezpečném stavu,
    - ★ jinak bude vlákno zablokováno a bude čekat na prostředek.

Předcházení vzniku uváznutí

- **Jak zjistit, že stav systému je bezpečný?**
  1. Existuje vlákno, které lze uspokojit volnými prostředky?
     - ⋆ Vlákno existuje
       ⇒ vlákno postupně uvolní všechny prostředky a opakujeme bod 1.
     - ⋆ Vlákno nexistuje
       ⇒ pokračujeme na bod 2.
  2. Byla uspokojena všechna vlákna?
     - ⋆ Ano ⇒ bezpečný stav.
     - ⋆ Ne ⇒ není to bezpečný stav.

Příklad: Bankéřův algoritmus

- **Je tento stav systému bezpečný?**

	R1	R2	R3
T1	3	2	6
T2	6	1	3
T3	3	1	4
T4	4	2	2

Request matrix Q

	R1	R2	R3
T1	1	0	0
T2	6	1	2
T3	2	1	1
T4	0	0	2

Allocation matrix A

R1	R2	R3
9	3	6

Existence vector E

R1	R2	R3
0	1	1

Free vector F

1. S volnými prostředky $F = [0, 1, 1]$ můžeme uspokojit vlákno T2.
   - Po určité době vlákno T2 skončí a uvolní své alokované prostředky.

	R1	R2	R3
T1	3	2	6
T2	6	1	3
T3	3	1	4
T4	4	2	2

Request matrix Q

	R1	R2	R3
T1	1	0	0
T2	6	1	2
T3	2	1	1
T4	0	0	2

Allocation matrix A

	R1	R2	R3
T1	2	2	6
T2	0	0	1
T3	1	0	3
T4	4	2	0

Missing matrix M

R1	R2	R3
9	3	6

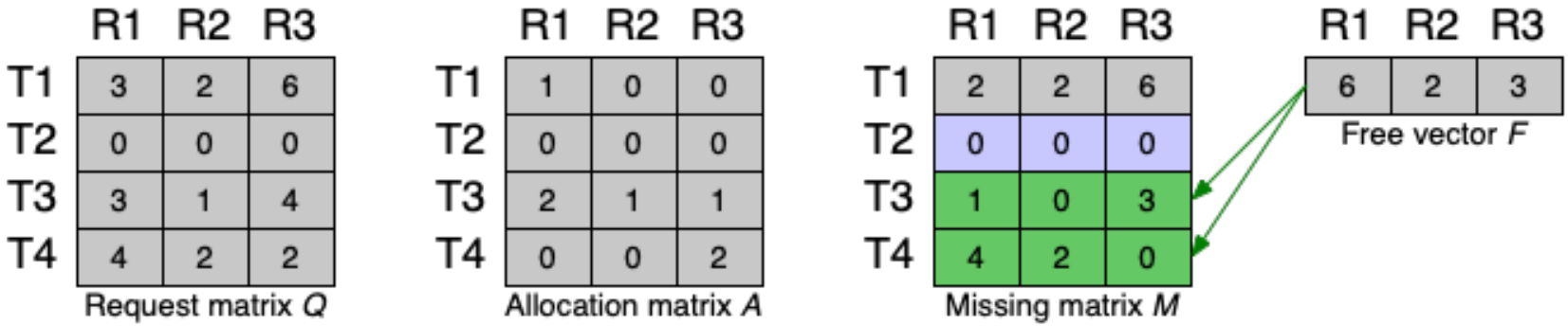
Existence vector E

R1	R2	R3
0	1	1

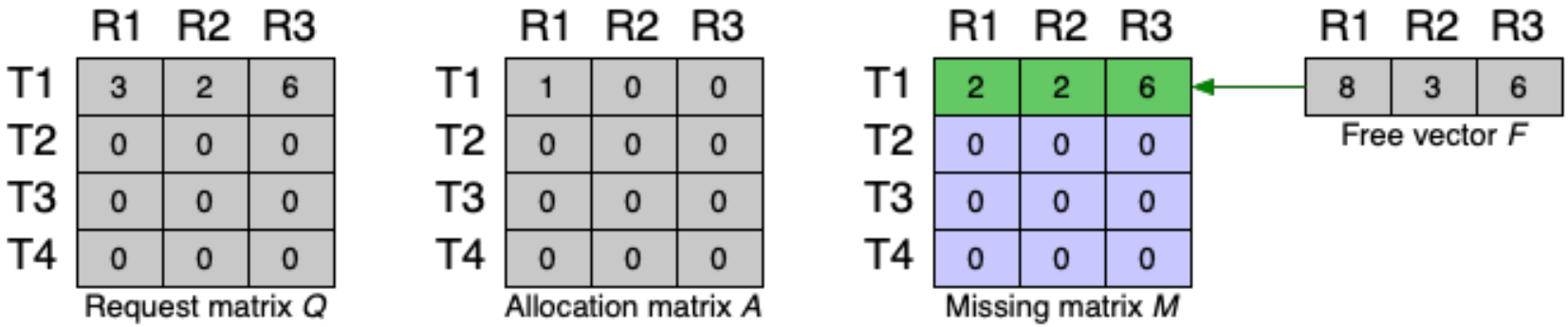
Free vector F

Příklad: Bankéřův algoritmus

2. S volnými prostředky $F = [6, 2, 3]$ můžeme postupně uspokojit vlákna T3 a T4.
   - Po určité době vlákna T3 a T4 skončí a uvolní své alokované prostředky.



3. S volnými prostředky $F = [8, 3, 6]$ můžeme uspokojit i poslední vlákno T1. $\Rightarrow$ systém je v bezpečném stavu.



Detekce a zotavení

- Předchozí strategie (prevence/předcházení uváznutí) jsou konzervativní a jsou založené na omezování přístup vláken k prostředků.

- Pokud však nemáme informace o tom, které prostředky budou vlákna používat nebo jak je budou používat, pak tyto strategie nelze použít $\Rightarrow$ k uváznutí může dojít.

- **Následující strategie s uváznutím počítá** a obsahuje dvě fáze.
  1. Detekci uváznutí
     - ⋆ V systému jsou prováděné pravidelné kontroly, které se snaží odhalit existující uváznutí.
  2. Zotavení z uváznutí
     - ⋆ Zotavení je založené na tom, že se část prostředků "uvolní" a tím se poruší čekání vláken ve smyčce $\Rightarrow$ uváznutí se odstraní.

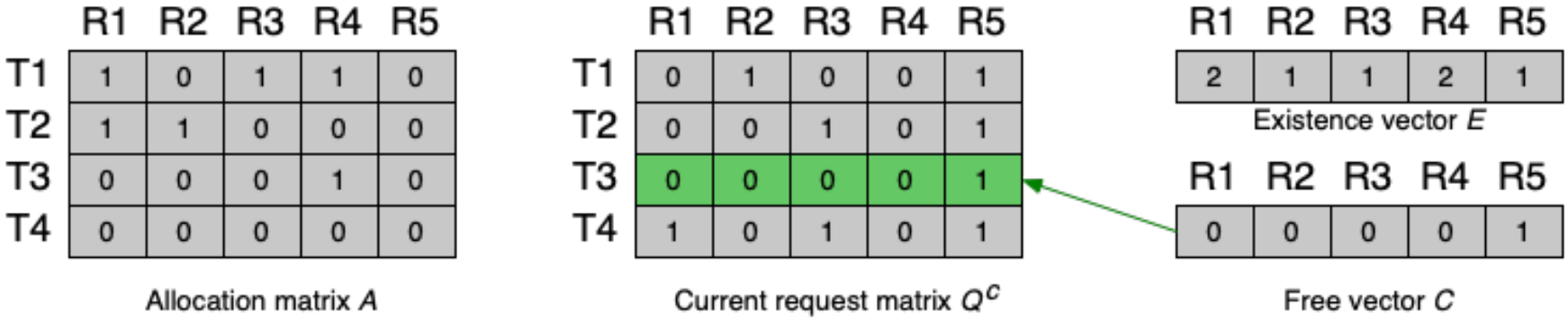
- Zatímco detekci lze implementovat poměrně jednoduše, samotné zotavení může být poměrně složité.

Detekce uváznutí

- **Popis aktuálního stavu systému**
  - vektor existujících prostředků E (existence vector),
  - vektor volných prostředků F (free vector),
  - matici přidělených prostředků A (allocation matrix),
  - matici požadavků Q^c (current request matrix),
    - ⋆ Obsahuje informaci o prostředcích aktuálně požadovaných jednotlivými vlákny.
- **Algoritmus pro detekci uváznutí**
  1. Vytvoříme kopii C vektoru F (aktuálně volné prostředky).
  2. Na začátku jsou všechna vlákna neoznačená.
  3. Označíme všechna vlákna, která nechtějí žádný prostředek.
  4. Existuje vlákno, které lze uspokojit prostředky z C?
     - ⋆ Vlákno existuje
       $\Rightarrow$ vlákno označíme, jeho alokované prostředky přičteme k vektoru C a opakujeme bod 4.
     - ⋆ Vlákno neexistuje
       $\Rightarrow$ pokračujeme na bod 5.
  5. Byla označena všechna vlákna?
     - ⋆ Ano $\Rightarrow$ k uváznutí nedošlo.
     - ⋆ Ne $\Rightarrow$ uváznutí nastalo (neoznačená vlákna jsou uvázlá).

Příklad: Detekce uváznutí

- **Nachází se v systému uváznutí?**



1. Volnými prostředky $C = [0, 0, 0, 0, 1]$ lze uspokojit vlákno T3.
   - Vlákno T3 označíme a jeho prostředky přičteme k vektoru C.
   - S prostředky $C = [0, 0, 0, 1, 1]$ nelze uspokojit žádný další proces
     $\Rightarrow$ vlákna T1, T2 a T4 jsou uvázlá.

Detekce uváznutí

- **Standardní nástroje OS pro detekci uváznutí**
  - Příkazy pro zobrazí stavu vláken (např. příkaz `ps`,...).
  - Příkazy, které dokáží zobrazit zásobníky vláken (např. příkaz `gstack`, debugger,...).
  - Různé aplikace pro ladění a profilování programů (např. Valgrind,...).

Zotavení z uváznutí

- **Ukončení všech uvázlých vláken**
  - Typické řešení v OS pokud uživatel/administrátor zjistí problém (např. pomocí zaslání signálu).
- **Postupné ukončování uvázlých vláken**
  - Postupně ukončujeme uvázlá vlákna dokud je v systému detekované uváznutí.
- **Zotavení pomocí návratu restartu**
  - Princip této strategie je založen na znovu spuštění uvázlých vláken z některého z předchozích stavů.
  - Toto řešení vyžaduje, aby byl v systému implementovaný mechanismus návratu (rollback) a opětovného spuštění (restart).
  - Systém si pravidelně ukládá svůj stav (důležité informace o systému) tak, aby později bylo možné obnovit systém do některého z předchozích stavů.
  - Díky nedeterminaci paralelního zpracování, je pravděpodobnost výskytu stejného uváznutí relativně malá.

Použité zdroje

1. A. S. Tanenbaum, H. Bos: *Modern Operating Systems (4th edition)*, Pearson, 2014.
2. W. Stallings: *Operating Systems: Internals and Design Principles (9th edition)*, Pearson, 2017.
3. A. Silberschatz, P. B. Galvin, G. Gagne: *Operating System Concepts (9th edition)*, Wiley, 2012.