

# Vzájemné vyloučení

---

B4B36PDV – Paralelní a distribuované výpočty

- Opakování z minulého cvičení
- Vzájemné vyloučení
- Zadání sedmé domácí úlohy

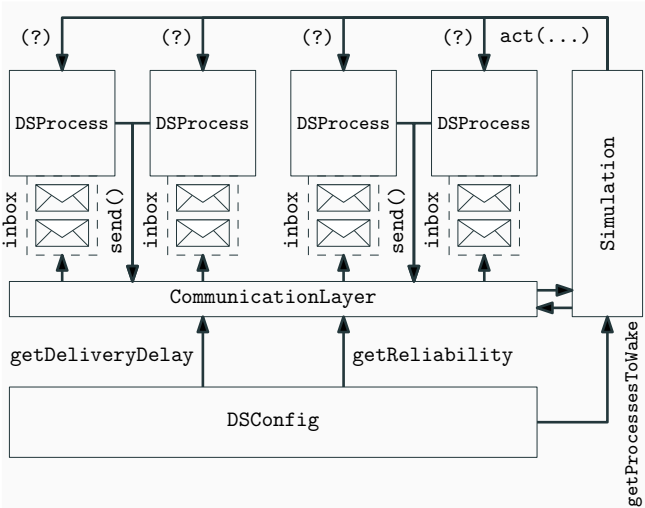
## Opakování z minulého cvičení

---



<http://goo.gl/a6BEMb>

# DSand framework



## Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas
2. mohou sloužit k detekci porušení kauzality
3. informují příjemce zprávy o hodinách odesílatele
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

## Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality
3. informují příjemce zprávy o hodinách odesílatele
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

## Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - **FALSE**
2. mohou sloužit k detekci porušení kauzality - **TRUE**
3. informují příjemce zprávy o hodinách odesílatele
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

Např. pokud komunikace dvou procesů na otázku  $T=5$  přijde odpověď s  $T=3$ , je jasné že je to odpověď na jinou otázku. Pro více procesů ale tato situace nemusí být zdetekována.



## Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality - TRUE
3. informují příjemce zprávy o hodinách odesílatele - TRUE
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

## Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - **FALSE**
2. mohou sloužit k detekci porušení kauzality - **TRUE**
3. informují příjemce zprávy o hodinách odesílatele - **TRUE**
4. vynucují totální uspořádání událostí v systému - **FALSE**
5. určují reálný čas, kdy byla zpráva poslána

Není to tak, např. zpráva 3 jednoho procesu může být odeslána před zprávou 1 od jiného procesu.

## Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - **FALSE**
2. mohou sloužit k detekci porušení kauzality - **TRUE**
3. informují příjemce zprávy o hodinách odesílatele - **TRUE**
4. vynucují totální uspořádání událostí v systému - **FALSE**
5. určují reálný čas, kdy byla zpráva poslána - **FALSE**

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - TRUE
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

Ano, vektorové hodiny správně zdetekují porušení kauzality vždy.

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - TRUE
3. generují částečné uspořádání zpráv - TRUE
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

Ano, všechny dvojice událostí jsou ve vztahu "nastalo po" nebo "je současné"

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - TRUE
3. generují částečné uspořádání zpráv - TRUE
4. určují reálný čas, kdy byla zpráva poslána - FALSE
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události



## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - **TRUE**
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - **TRUE**
3. generují částečné uspořádání zpráv - **TRUE**
4. určují reálný čas, kdy byla zpráva poslána - **FALSE**
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události - **FALSE**

Hodiny nedokáží detekovat kauzalitu (následek nastal v důsledku příčiny), ale dokáží detekovat potenciální kauzalitu (následek mohl nastat v důsledku příčiny, tzn. existuje kauzální cesta od příčiny k následku)

## Vzájemné vyloučení

---

S přístupem více vláken k **jednomu zdroji** jsme se již setkali

→ Musíme zaručit konzistenci zdroje

Např. v **OpenMp** pomocí

`#pragma omp critical`

S přístupem více vláken k **jednomu zdroji** jsme se již setkali

→ Musíme zaručit konzistenci zdroje

Např. v **OpenMp** pomocí `#pragma omp critical`

Jak to vyřešit v případě DS?

*Nejjednodušší možnost:* O zdroj se stará samostatný proces

→ Běží na samostatném stroji

→ Může použít vlastní způsoby synchronizace

*Nejjednodušší možnost:* O zdroj se stará samostatný proces

→ Běží na samostatném stroji

→ Může použít vlastní způsoby synchronizace

---

V čem je tedy problém?

*Nejjednodušší možnost:* O zdroj se stará samostatný proces

→ Běží na samostatném stroji

→ Může použít vlastní způsoby synchronizace

---

## V čem je tedy problém?

Některé praktické případy DS toto **neumožňují**

- Požadujeme bezstavovost zdroje  
(souborové NFS servery)
- Zdroj nemá výpočetní jednotku  
(sítě Ethernet a IEEE 802.11, procesy přistupují k jednomu výstupnímu komunikačnímu kanálu)
- ... a jiné

U procesů máme podobné požadavky jako u vláken



U procesů máme podobné požadavky jako u vláken

- **Safety:** v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces

U procesů máme podobné požadavky jako u vláken

- **Safety:** v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
- **Liveness:** každá žádost o přístup ke zdroji je splněna v konečném čase

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
- **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase
- **Fairness**: procesy získávají přístup k pořadí, v jakém o něj požádali

---

A hodnotíme je podobným způsobem

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
  - **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase
  - **Fairness**: procesy získávají přístup k pořadí, v jakém o něj požádali
- 

A hodnotíme je podobným způsobem

- Kolik zpráv je nutné si vyměnit, aby došlo k získání a poté uvolnění zdroje?

U procesů máme podobné požadavky jako u vláken

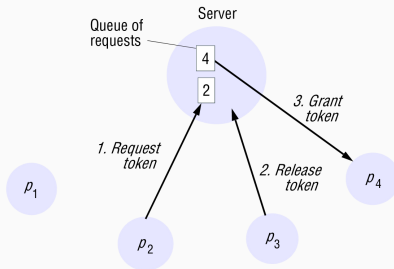
- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
  - **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase
  - **Fairness**: procesy získávají přístup k pořadí, v jakém o něj požádali
- 

A hodnotíme je podobným způsobem

- Kolik zpráv je nutné si vyměnit, aby došlo k získání a poté uvolnění zdroje?
- Kdy nejdříve po uvolnění může zdroj získat další proces?

Jaké možnosti tedy v DS máme?

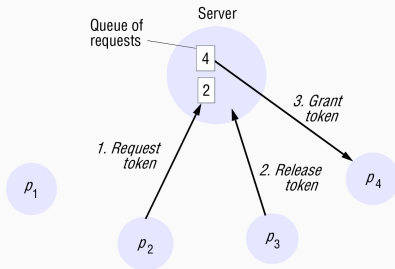
Jeden z procesů je určený jako správce požadavků



Udržuje si frontu doručených požadavků

Přiznává přístup ke zdroji v pořadí daném frontou

Jeden z procesů je určený jako správce požadavků



Udržuje si frontu doručených požadavků

Přiznává přístup ke zdroji v pořadí daném frontou

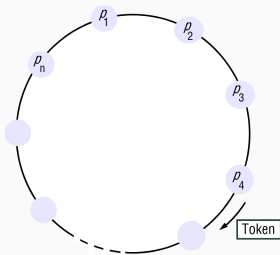
:( To jsme si moc nepomohli (zavedli jsme single point of failure)

Navíc není splněn požadavek o zachování pořadí (pořadí závisí na latenci komunikace)



# Kruhové splňování

Procesy jsou uspořádané v kruhu



Posílají si povolení k přístupu ke zdroji

Jakmile proces zdroj již nepotřebuje, pošle povolení dál

Opět není splněn požadavek o zachování pořadí

Co použít nějakou techniku kterou již známe?

Co použít nějakou techniku kterou již známe?

Serializaci jsme v DS již využívali...

Co použít nějakou techniku kterou již známe?


Serializaci jsme v DS již využívali...

**Hodiny!**

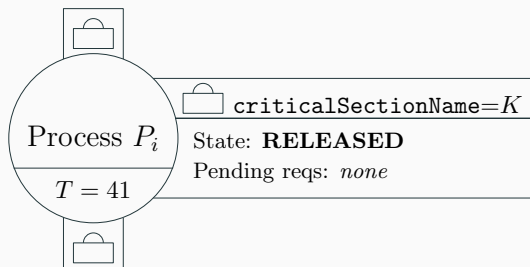
---

Jak je tedy konkrétně použít?

# Ricart-Agrawalovo vyloučení

 Ricart, Agrawal: An optimal Algorithm for Mutual Exclusion in Computer Networks, 1981

## Žádost o vstup do kritické sekce



$P_j$

$P_k$

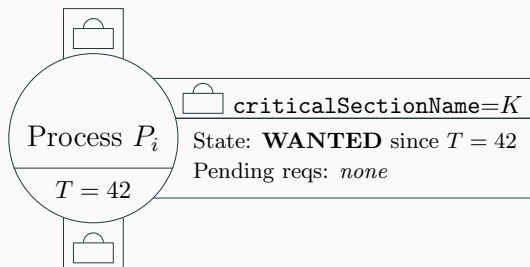
$P_l$

## Žádost o vstup do kritické sekce



1. Pokud chce proces  $P_i$  požádat o vstup do kritické sekce  $K$ , zaznamená čas  $T_i$  kdy o zdroj žádá a pošle zprávu  $REQUEST(K)$  s tímto časem všem procesům, které do  $K$  přistupují. Nastaví stav zámku na **WANTED**.

## Žádost o vstup do kritické sekce



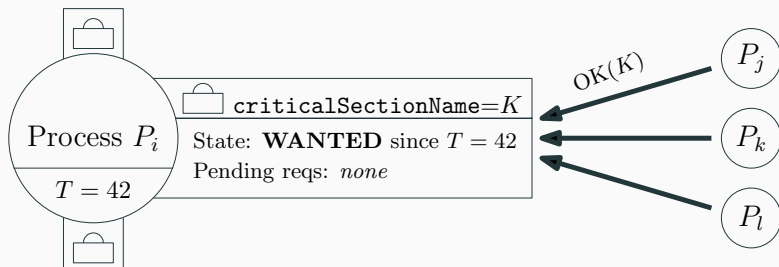
$P_j$

$P_k$

$P_l$









- 
2. Zámek  $K$  procesu je ve stavu **WANTED** dokud neobdrží zprávu  $OK(K)$  od každého dalšího přístupujícího procesu. Poté se nastaví na **HELD**.



## Příchozí požadavek od jiného procesu

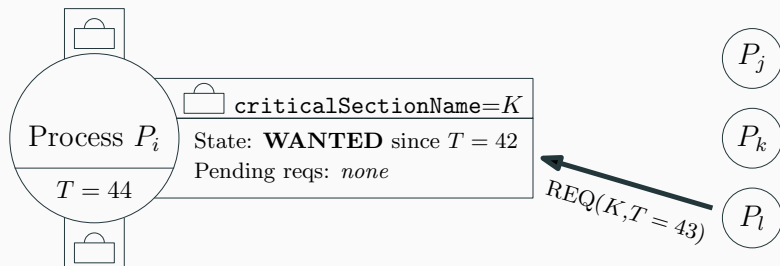




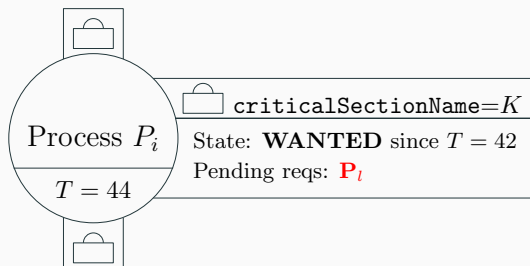


- 
3. Pokud procesu  $P_i$  přijde zpráva REQUEST( $K$ ) od procesu  $P_j$  s časem  $T_j$ :
- (i) pokud je zámek  $K$  ve stavu **RELEASED**, nebo je ve stavu **WANTED** a o vstup do kritické sekce žádal v čase  $T_i > T_j$ , pak pošle zprávu  $OK(K)$  procesu  $P_j$ ,

## Příchozí požadavek od jiného procesu



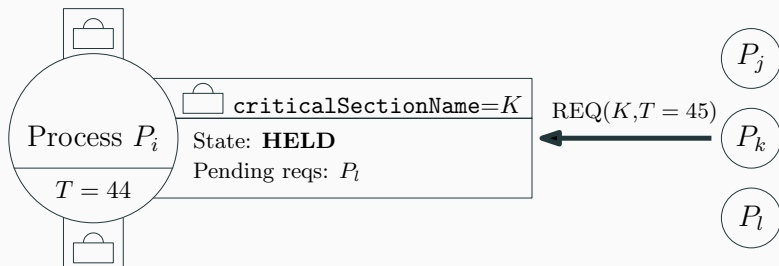


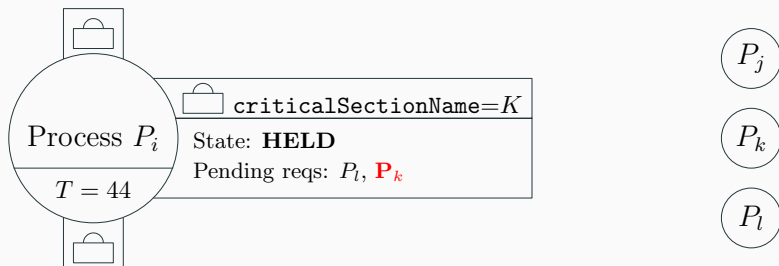


$P_j$

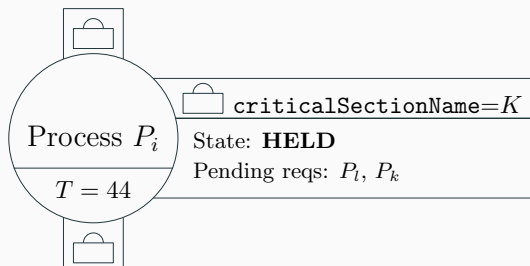
$P_k$

$P_l$





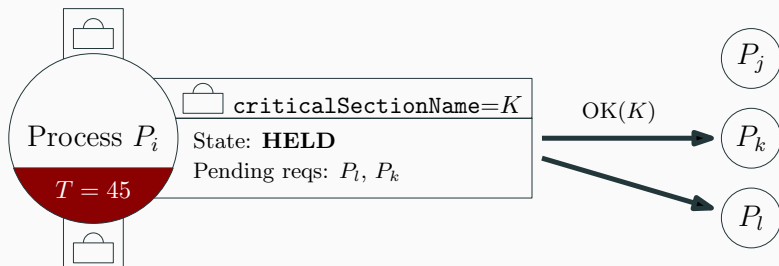
- 
3. Pokud procesu  $P_i$  přijde zpráva REQUEST( $K$ ) od procesu  $P_j$  s časem  $T_j$ :
- (ii) jinak požadavek odloží a neodpoví.



$P_j$

$P_k$

$P_l$





- 
4. Pokud proces  $P_i$  dokončí práci v kritické sekci  $K$ , nastaví stav zámku  $K$  na **RELEASED**, odpoví na všechny odložené požadavky a frontu požadavků vyprázdní.

Jaké požadavky tento algoritmus splňuje?

1. Safety?
2. Liveness?
3. Fairness?
4. Počet zpráv?



<http://goo.gl/a6BEMb>

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness?
3. Fairness?
4. Počet zpráv?



Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness? TRUE
3. Fairness?
4. Počet zpráv?

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness? TRUE
3. Fairness? TRUE
4. Počet zpráv?

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness? TRUE
3. Fairness? TRUE
4. Počet zpráv?  $2(n - 1)$

Dokáží se algoritmy vypořádat se ztrátou dat?

Dokáží se algoritmy vypořádat se ztrátou dat?

---

Dokáží se vypořádat s padajícími procesy?

Dokáží se algoritmy vypořádat se ztrátou dat?

---

Dokáží se vypořádat s padajícími procesy?

**⚠** Požadavkem algoritmu je spolehlivá komunikace. Selhávající procesy mohou vést k zamrznutí (bez dodatečných úprav).

## Zadání samostatné úlohy

---

## Doprogramujte Ricart-Agrawalův algoritmus

Doimplementujte logiku Ricart-Agrawalova algoritmu ve třídě `exclusion/ExclusionPrimitive.java`. Následně spusťte scénář `bank.Main`.

**⚠** Implementace tohoto zadání je obsahem 7. domácího úkolu s termínem odevzdání 10. 5. 2020!

Zpracování musí být **distribuované**, procesy si nesahají vzájemně do paměti!



Díky za pozornost!

Budeme rádi za Vaši  
zpětnou vazbu! →



<https://bit.ly/3f3bNtk>