

## 7. domácí úloha: Ricart-Agrawalovo vyloučení

Vzájemné vyloučení (anglicky mutual exclusion, nebo zkráceně mutex) je algoritmus používaný v konkurentním programování jako synchronizační prostředek. V paralelní části předmětu PDV jsme s mutexy pracovali prakticky na každém cvičení. Mutex zabraňuje tomu, aby dvě vlákna (nebo procesy) vykonávala operace nad stejným sdíleným prostředkem - aby současně vstoupila do stejné kritické sekce. V paralelním programování za nás problém vzájemného vyloučení řešil operační systém, který plnil roli centrální autority. Na rozdíl od tohoto centralizovaného řešení se v distribuovaných výpočtech musí všechny procesy shodnout na tom, kdo v dané chvíli bude mutex vlastnit.

V tomto domácím úkolu si vyzkoušíte implementaci jednoho z algoritmů pro vzájemné vyloučení v distribuovaném prostředí. V takovém systému obecně nelze využít prostředků poskytovaných jediným lokálním strojem, řešení musí být od základu postaveno pouze na výměně zpráv. Algoritmus, který budete implementovat, se podle svých tvůrců jmenuje Ricart-Agrawalův a funguje následujícím způsobem:

Každý proces má své lokální skalární logické hodiny, a zámek (či několik zámků), kde každý má

1. identifikátor kritické sekce, kterou zamyká,
2. stav: *RELEASED*, *HELD* nebo *WANTED*, a
3. frontu odložených požadavků.

Na začátku je stav každého zámku každého procesu nastaven na *RELEASED*. V systému kolují pro každou kritickou sekci dva typy zpráv: *REQUEST* a *OK*

1. Pokud chce proces  $P_i$  požádat o vstup do kritické sekce  $K$ , zaznamenaná čas  $T_i$  kdy o zdroj žádá a pošle zprávu *REQUEST*( $K$ ) s tímto časem všem procesům, které do  $K$  přistupují. Nastaví stav svého zámku  $K$  na *WANTED*.
2. Zámek  $K$  procesu je ve stavu *WANTED* dokud neobdrží zprávu *OK*( $K$ ) od každého dalšího přistupujícího procesu. Poté se nastaví na *HELD*.
3. Pokud procesu  $P_j$  přijde zpráva *REQUEST*( $K$ ) od procesu  $P_i$  s časem  $T_i$ , tak
  - pokud je zámek  $K$  ve stavu *HELD*, pak zprávu *REQUEST*( $K$ ) zařadí mezi odložené požadavky a neodpoví
  - pokud je ve stavu *WANTED* a o vstup do kritické sekce žádal v čase  $T_j < T_i$ , případně  $T_j = T_i$  a  $j < i$ , pak zprávu *REQUEST*( $K$ ) zařadí mezi odložené požadavky a neodpoví,
  - jinak pošle zprávu *OK*( $K$ ) procesu  $P_i$ .

4. Pokud proces  $P_i$  dokončí práci v kritické sekci K, nastaví stav zámku K na *RELEASED*, odpoví na všechny zprávy ve frontě zámku a frontu vyprázdní.

Vaším úkolem bude doimplementovat třídu `exclusion.ExclusionPrimitive` v `hw08_exclusion.zip` [[https://cw.fel.cvut.cz/wiki/\\_media/courses/b4b36pdv/tutorials/hw08\\_exclusion.zip](https://cw.fel.cvut.cz/wiki/_media/courses/b4b36pdv/tutorials/hw08_exclusion.zip)], která bude implementovat zámek podle protokolu Ricart-Agrawala. Instanci této třídy vlastní každý proces, který chce přistupovat do kritické sekce se jménem `criticalSectionName` (seznam všech procesů, které rozhodují o přístupu do kritické sekce naleznete v poli `allAccessingProcesses`). Proces, který danou instanci `ExclusionPrimitive` vlastní na ní může volat následující metody:

- `requestEnter()` ve chvíli, kdy chce vstoupit do kritické sekce se jménem `criticalSectionName`
- `isHeld()` pro zjištění, zda mu byl už přidělen exkluzivní přístup do sekce `criticalSectionName`
- `exit()` pro opuštění kritické sekce `criticalSectionName` (a uvolnění zdroje pro ostatní procesy)

`ExclusionPrimitive` zpracovává zprávy pomocí metody `accept()`. Pokud se zpráva týká kritické sekce `criticalSectionName`, tak ji zpracujte a vraťte `true`. V opačném případě vraťte `false`. Ve Vašem kódu můžete používat metody procesu, který danou instanci `ExclusionPrimitive` vlastní - metody objektu `owner`.

V aplikaci může být (a bude) více kritických sekcí s různými `criticalSectionName`. Při zpracovávání zpráv tak musíte ověřovat, zda se přijatá zpráva skutečně týká kritické sekce, kterou obsluhuje daný `ExclusionPrimitive`!

Abychom Vám usnadnili práci, naimplementovali jsme Vám logiku Lamportových (skalárních) hodin. Tato logika je implementovaná ve třídě `clocked.ClockedProcess` a zajišťuje, že:

- Po přijetí zprávy dojde k aktualizaci logického času pomocí pravidla  $\text{currentTime} = \max\{\text{currentTime}, \text{msg.sentOn}\} + 1$
- Každá odeslaná zpráva je označena aktuálním logickým časem odeslání ( $\text{msg.sentOn} = \text{currentTime}$ )

Inkrementaci logického času (například před odesláním zprávy) musíte provádět ručně voláním metody `increaseTime()`.

Abyste mohli využít `ClockedProcess` u, Vaše zprávy musí dědit od třídy `clocked.ClockedMessage` (a nikoliv od obecné třídy `Message`)!

Vaší implementaci můžete vyzkoušet na scénáři `bank.Main`. Několik bankovních úředníků/úřednic ( `BankOfficerProcess` ) zpracovává příkazy k převodu mezi bankovními účty. Ve chvíli, kdy dochází ke zpracování příkazu pro převod peněz z účtu `accounti` na účet `accountj` dojde

1. Ke vstupu do kritických sekcí `accounti` a `accountj` (voláním metody `requestEnter` odpovídajícího objektu `ExclusionPrimitive` a následným vyčkáním na splnění podmínky `isHeld()==true`). Vzpomeňte si, že jednou z možností, jak lze v případě zamykání více mutexů předejít deadlocku, je zamykat je v přesně daném pořadí - to zajišťuje třída `MultiLock`.
2. K přečtení aktuálních hodnot účtů `i` a `j` zasláním zprávy `ReadMessage` procesu `DatastoreProcess` a vyčkáním na odpověď (zprávy `ValueMessage`).
3. K zapsání nových zůstatků účtů `i` a `j` zasláním zprávy `WriteMessage` procesu `DatastoreProcess` a vyčkáním na potvrzení (zprávami `WriteAcknowledgedMessage`).
4. K opuštění kritických sekcí `accounti` a `accountj` voláním metody `exit()` na odpovídajících instancích `ExclusionPrimitive`.

Zazipovaný soubor `ExclusionPrimitive.java` s případnými vlastními třídami (nesmí přepisovat kód projektu) odevzdávejte do systému BRUTE do **11.5. 23:59 CET pro středeční cvičení** a **19.5. 23:59 CET pro čtvrteční cvičení**.

Za úlohu můžete získat maximálně **2 body** v závislosti na kvalitě vašeho řešení.

Pokud využíváte vlastní třídy, musí se tyto třídy nacházet v package `exclusion` !

courses/b4b36pdpv/tutorials/hw\_07.txt · Last modified: 2023/03/02 00:56 by macejp1