**Projekt NPR**

Założenia:

1. Centralny host (portMapper), odpowiedzialny za informowanie o aktualnym składzie grupy
2. Hosty nie ulegają awarii (ale mogę się odłączać?)
3. Każdy proces ma swój serwer komunikacyjny

Zadania portMappera:

1. Przechowuje informacje o aktualnym składzie przetwarzania
2. Przyjmuje wiadomości od nowo dołączających się hostów, odsyła info o adresach aktywnych hostów (req/rep)
3. Informuje hosty o dołączeniu/odłączeniu się nowego hosta (pub-sub)

**PortMapper**

**PMPU1:**

1. Przyjmuje wiadomość na gnieździe REP od nowego hosta (razem ze swoim adresem pub-sub)
2. Odsyła wiadomość do nowego hosta z adresami aktywnych hostów w systemie. (wraz z adresami pub-sub wszystkich serwerów i **Portmappera** ). **Nadaje procesowi ogólne i unikalne id.**
3. Przyjmuje wiadomość na gnieździe REP o gotowości do przetwarzania przez nowy host
4. Rozsyła wiadomość do wszystkich hostów informującą o dołączeniu do przetwarzania nowego hosta (w wiadomości adres nowego hosta)

**Zajmowanie sekcji krytycznej (wywołanie Monitor.wait()):**

1. Wywołanie wait() powoduje rozesłanie wiadomości o chęci wejścia do sekcji krytycznej (pub-sub?)
2. Stosujemy tutaj algorytm Rikarda-Agawy do uzyskania rozproszonego dostępu do sekcji krytycznej
3. Proces może kontynuować przetwarzanie po otrzymaniu odpowiedzi od wszystkich hostów.

**Tematy do przypomnienia:**

-> algorytm Rikarda-Agrawali (synchronized w Javie takie lock - unlock)

->

**Tematy do pomyślenia:**

-> czy robimy tylko lock-unlock, czy wait-notify (czy oba)

* Propozycja: gdy robimy wait() nie powiadamiamy nikogo o tym (inkrementujemy zegar), gdy ktoś robi notify, to wysyła do wszystkich i oni muszą między sobą uzgodnić kto ma dostać notify (Agrawali - o najmniejszym zegarze). Jeżeli notifyAll, to wysyłamy do wszystkich i nie ma uzgadniania.
* porównanie zdarzeń: zegar matterna + id procesu

-> jaki schemat komunikacji do powyższych (req-rep, pub-sub, push-pull)

* req-rep na początku a potem wszędzie pub-sub bo agrawali (Jeżeli będzie trzeba zrobić semafor o pojemności 10 to wtedy agrawali: n -1 - b)

[*https://stackoverflow.com/questions/17814436/difference-between-pub-sub-and-push-pull-pattern-in-zeromq*](https://stackoverflow.com/questions/17814436/difference-between-pub-sub-and-push-pull-pattern-in-zeromq)

-> jak przesyłamy dane (JSON vs. Protobuf)

* protobuf <https://www.baeldung.com/google-protocol-buffer>

-> błędy komunikacyjne, awarie hostów - czy na pewno pomijamy

* pomijamy

-> co trzeba zabić na sztywno (czy na pewno tylko adres REP centralnego portMappera)

* każdy ma adres REP portmappera, adres na którym ma stać apka, port

-> jak rozróżniać rozproszone obiekty-monitory (id przez programistę, jakoś inaczej)

* programista sam nadaje id obiektu

-> co robimy docelowo (skala projektu): rozproszone analogi z java.util.concurrent? gotowe rozwiązania dla problemów, np. producent-konsument? *research - co się da w sensownym czasie*

* *do ustalenia później*

-> Jak ma wyglądać struktura wiadomości ???

*{zegar, idProcesu, typ, idObiektu}*

*typ: notify, notifyAll, lock, unlock*

-------------------------------------------------------------------------------

Ustalenia 09.09.2021

Obiekt.lock() -> wysyłamy LOCK\_REQ, czekamy na LOCK\_ACK od wszystkich

Obiekt.unlock() -> wysyłamy LOCK\_ACK do wszytskich czekających (zgodnie z algorytmem Agawy)

Obiekt.wait() -> pusto, czekamy na NOTIFY albo NOTIFY\_ALL

Obiekt.notify() -> wysłamy NOTIFY

Obiekt.notifyAll() -> wysłamy NOTIFY\_ALL

Po odebraniu NOTIFY\_ALL - easy

Po odebraniu NOTIFY:

Jeśli mieliśmy zrobionego waita - rozpoczynamy sesję Agrawali wiadomościami NOTIFY\_REQ i NOTIFY\_ACK, powiązaną z konkretnym notifyID. Po odebraniu NOTIFY\_ACK od wszystkich pozostałych rozsyłamy NOTIFY\_RST, kończąc sesję powiązaną z konkretnym notifyID.

Jeżeli w trakcie sesji NOTIFY\_REQ/ACK przyjdzie kolejne NOTIFY, to możemy:

-> od razu wysłać NOTIFY\_REQ, jeśli nie obiegamy się o dostęp w poprzedniej sesji

-> poczekać na zakończenie poprzedniej sesji i po przegranej obiegać się o dostęp w nowej

Jeżeli w InitRequestMessage jest ready na true to wiadomo, że port mapper może rozesłać innym procesom informację o naszej gotowości. Generalnie wiadomość InitRequestMessage bedzie wysyłana dwukrotnie. Raz na początku z ready na false, a potem na true gdy już proces wysyłający podepnie się pod wszystkie inne kolejki MQ (będzie po prostu gotowy).

