


49. DISTRIBUOVANÝ BROADCAST A SYNCHRONIZACE V DISTRIBUOVANÝCH SYSTÉMECH

Broadcast - nějaký uzel / procesor rozšíří zprávu na všechny ostatní podle topologie
 - každá zpráva obsahuje sekvenční číslo a odesílatele (Lamport) 

Synchronizace - porovnání uspořádání mezi jednotlivými procesy / synchronizací
 - a různými systémy zjednotí se sdílenou pamětí lze povolit semafor, monitor, ...
 - a systémech které jsou distribuované ale mají ~~společný~~ ^{paralelní} společný prostředek lze realizovat pomocí systému ADA či LINDA
 - a systémech které jsou distribuované a nemají žádný sdílený prostředek ale pro synchronizaci si procesy musí najít nějaký způsob jak se navzájem sdílet
 - jsou tři způsoby: 1) synchronizace globální (reálným) časem
 2) synchronizace master uzlem
 3) synchronizace založená na shodě mezi procesy

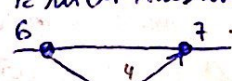
1) Synchronizace globální (reálným) časem
 • Berkeley algorithms
 • NTP protocol (Network Time Protocol) - např. přes UDP rozšířením čas na multicast
 • Marzull algorithms


2) Synchronizace master uzlem
 - jak určit který uzel bude master a bude synchronizovat všechny ostatní?
 ⇒ algoritmus Chang and Roberts
 - každý proces (uzel) má unikátní UID → vybere se nejmenší
 - toto UID se každé době posílá odtí a pokud uzel má vyšší UID než přijde tak posílá svoje číslo to co má přišlo
 - se dřívešním bude se už někdo posílá jen to nejmenší

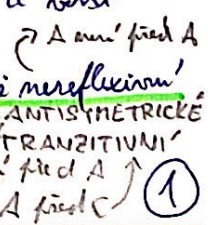
3) Synchronizace založená na shodě procesů
 algoritmy založené na časových razítkách: Lamportův algoritmus
 + optimalizace pomocí RA
 algoritmy založené na tokenech: Raymondův stromový algoritmus

Lamportovy hodiny - happened-before - reflexivní, částečně uspořádané, transitivity
 - neměřivá se čím globální (fyzikální / reálný) čas čím řádový master uzel
 - každý uzel / proces má svoje vlastní logické hodiny, které běžně ručně
 - tyto hodiny si vždy navzájem synchronizují pravidelným sporem - ale jen na chvíli
 - založeno na relaci happened-before

↳ spora byla nejprve desítková a až potom desítková
 - proces posílá sporu se sporem i aktuální stav svých hodin jako časové razítko
 - druhý proces přijme a porovná hodinu svých hodin a přičítá časové razítko a vrátí
 se mičkou nastavená jako své hodiny a inkrementuje (+1)

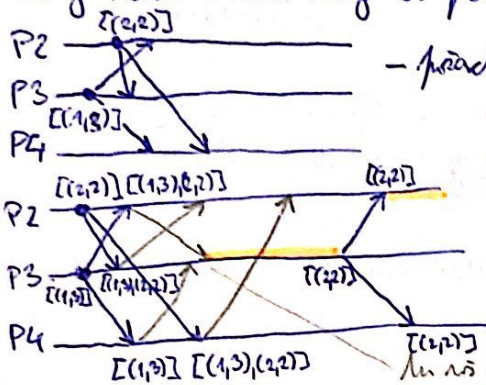
 - 4 < 6 a tak nechá svoje a jen inkrementoval 6++

 - 6 > 4 a tak nastavil své hodiny na 6 a inkrementoval

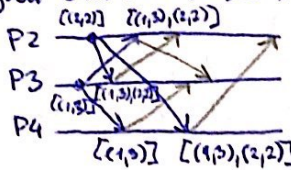
! ⇒ relace happened-before je reflexivní
částečně uspořádané
 ANTISYMETRICKÉ
 TRANZITIVNÍ
 1) 

Lamportův algoritmus

- algoritmus pro řešení přístupu do KS mezi distribuovanými uzly pomocí samostatné správy
- realizován na FIFO poslání správy a chování
- každý proces udržuje lokální prioritní frontu a řádchů do KS - předchozí (prioritní číslo č. procesu)
- ve frontách jsou řádky seřazené podle časových značek
- pokud proces chce do KS tak do své fronty vloží číslo a svou frontu pošle všem ostatním procesům
- každý proces upraví svou frontu podle toho co mu přišlo a pošle odpovíď se svou frontou
- proces co chce do KS si pošle na všechny odpovídi a podle nich aktualizuje svou frontu
- pokud je po přijetí a aktualizaci první ve frontě tak odpovíď do KS jinak čeka
- na konci KS se vyjme z fronty a pošle všem ostatním procesům svou frontu a j
- si ji zase aktualizuje a pokud jsou oni na místě tak odpovíď do KS atd...



- předchozí



ODPOVĚDI

- každý z odpovídek nám co aktualizoval - vložil na místo aktualizací fronty

- KS a volání KS - odpovíď nám pak P2

anť má P3 všechny odpovídi a je první ve frontě tak odpovíď do KS

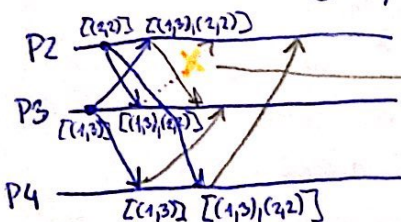
- 1 proces chce do KS $\Rightarrow (n-1)$ žádostí + $(n-1)$ odpovídek + $(n-1)$ volání

máme počet procesů (uzlů)

\rightarrow to je hodně moc : (- řešení RICARD-AGRAWALA ALGORITHMUS

Ricard - Agrawala algoritmus (RA)

- optimalizace Lamportova algoritmu - řeší se Lamport a RA optimalizací
- snižuje počet zpráv z $3 \cdot (n-1)$ na $2 \cdot (n-1)$
- pokud přijde požadavek a proces si je daný proces o předchozím je to prioritní frontě co se má tak mu nepošle odpovíď ale pošle mu až volání až vyjde z KS \Rightarrow slučuje správy voláním a ~~odpovídi~~ odpovídi



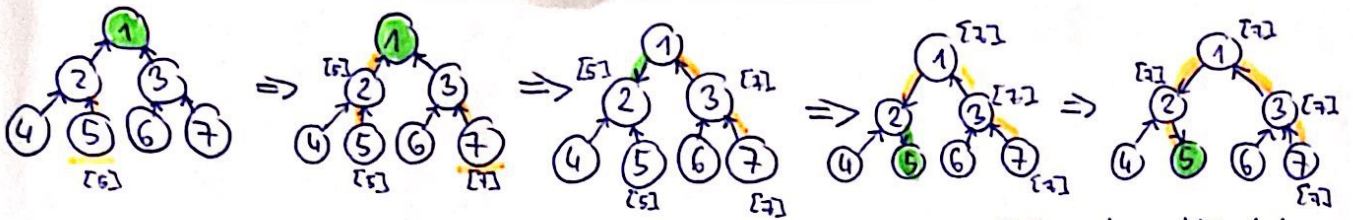
tu nepošle odpovíď, a pošle až volání

Raymondův stromový algoritmus

- algoritmus realizovaný na hořem, kde má hořem snižuje do KS, na začátku jím má hořem
- a všechny uzly udržují na rootu - je to strom
- procesy v uzlech řídí hořem a hořem - do uzlu po cestě sápnutí svoje ID až dojde do hořem - hořem pak pošle doho stromem ten hořem a malí odpovíď do KS - pokud řídí dále tak pošle hořem a jde po tom ^{drůb} do uzlu až řídí hořem kde ho má - atd...

\Rightarrow hořem se orientují ten hořem řídí hořem

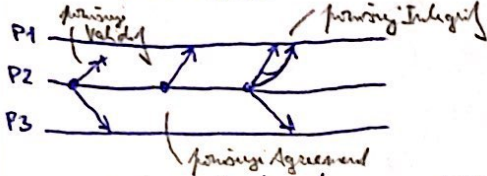
(2)



- až 5 rozjedu a KS tak se bude předávat ⑦

Broadcast

- spolehlivý broadcast = **RELIABLE**



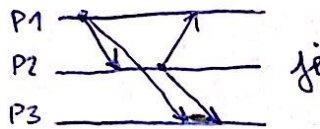
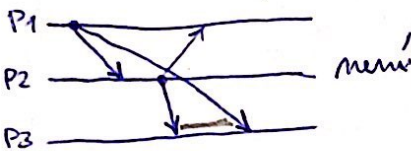
PLATNOST

Validity - pokud byla správná nějakým procesem obdržena
 +
Agreement - pokud proces obdržel správnou tak i ostatní
 +
Integrity - každý správně je obdržena jednotně (nemí smíchání)

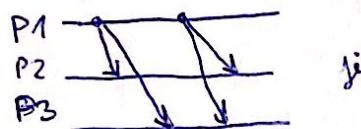
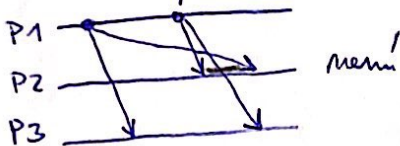
- další vlastnosti

FIFO order / uspořádání - P přijde m a pak n tak všem musím přijít m
Causal order - P2 obdržel m od P1 a pak přijde n tak n musí přijít
Kausalní uspořádání - P2 obdržel m od P1 a pak přijde n tak n musí přijít
Total order - P1 obdržel m a P2 obdržel n tak všechny procesy
Úplné uspořádání - obdržel buď m, n nebo n, m - žádná je jistě m
 principiálně ale všechno se skládá

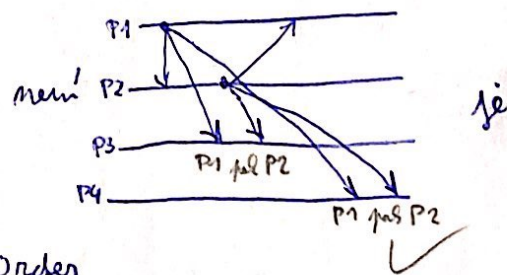
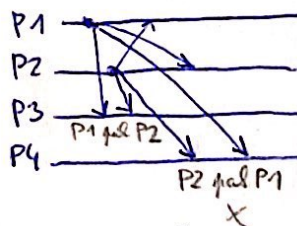
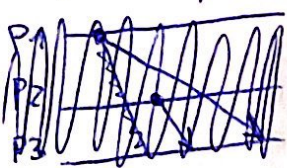
Kausalní uspořádání (Causal Order)



FIFO uspořádání (FIFO Order)



Topořádkání / úplné uspořádání (Total Order)



FIFO Broadcast = FBCAST = Reliable + FIFO Order

Causal Broadcast = CBCAST = Reliable + Causal Order

Atomic Broadcast = ABCAST = Reliable + Total Order

Pozn.: v komputačním alg. proces P2 přijde v čase 2 předtím než P3 přijde v čase 1 jako proces P3 tak máme frontu [(1,2), (1,3)] a zde předchází mu nějaký případ kdy relace happened-before byla narušena a
 vytvořila se \Rightarrow abychom obnovili úplné uspořádání tak se v případě shody čísel
 používá jako prioritní číslo procesu - máme-li přednost