# PA150 19.1. 2014

## TRANSAKCE

### ACID A KDO TO ZAJIŠŤUJE

- **A – atomicity** (provede se pouze celá transakce nebo vůbec, bez ohledu na souběžnost transakcí a výpadky). Na disku je uložen deník log s informacemi o zápisu (id, hodnota před a po), případně se odtud obnoví původní hodnoty, rollback – násilné zrušení transakce a návrat do původního stavu (kompenzační transakce). Zajišťuje to databázový systém, TPM (správa obnovy).
- **C – consistency** (koresponduje s údaji z reálného světa, provede se validně, nenaruší konzistenci báze dat). Ověřují se integritní podmínky před a po. Zajišťuje to programátor, případně automatické testy.
- **I – isolation/independence** (transakce probíhají odděleně sekvenčním zpracováním, paralelním nebo díky plánu, neovliňují se, stejný výsledek jako postupně). Funkční složka DBS nebo TMP (systém řízení souběžnosti, koordinátor transakce).
- **D – durability** (trvanlivost – neztrácí se data, výsledky transakce jsou trvalé). Modifikace uloženy na disk před dokončením transakce nebo je dostatečné info o modifikacích prováděných transakcí. Zajišťuje TPM (správa obnovy).

### KASKÁDNÍ VRACENÍ

- **Kaskádní vracení** – zkrachování jedné transakce může způsobit i další, je to zatěžující, vyhýbat se tomu (i když omezujeme souběžnost)! Nedojde k němu, když je t_commit Ti před read Tj.

### PRINCIP KONTROLNÍCH BODŮ

- **Checkpoint** – kontrolní bod pro zkrácení obnovy, při vytváření se provede výstup deníkových záznamů v RAM na disk, výstup všech modifikovaných bloků na disk, výstup záznamu <checkpoint L> do deníku, L – seznam aktivních transakcí v momentě. Během vytváření nesmí transakce operovat s daty nebo zapisovat do deníku. Obnovují se pak jen transakce neprovedené po bodě nemající commit v L. Transakce mající v deníku <commit> nebo <abort> se provede redo, jinak undo.

### SERIALIZOVATELNÝ PLÁN, OBNOVITELNÝ A SÉRIOVÝ PLÁN

- **Serializovatelný plán**: určuje pořadí všech transakcí řešených souběžně, zachová pořadí instrukcí, ve kterém se instrukce vyskytují v jednotlivých transakcích a zajišťuje, aby výsledek provedených transakcí byl validní.
- **Obnovitelný plán** respektuje omezení zajišťující zachování obnovitelnosti báze dat. Platí, pokud Tj lze prohlásit za provedenou (t_commmit) až po prohlášení Ti za provedenou.
- **sériový plán** – má instrukce transakcí jako kontinuální bloky, možných plánů je n!
- **nesériový** – transakce se křekrývají, případně jsou po částech prokládané, >>n!, může být správný (pak je serializovatelný)
- Operace Oi, Oj, transakce Ti, Tj. Operace jsou konfliktní <=> přistupují ke stejné datové položce a alespoň jedna z nich je operace zápisu. Dva plány jsou ekvivalentní, pokud se liší pouze pořadím provedení nekonfliktních operaci.

## OBNOVA

### OBNOVA PODLE DENÍKU PŘI OKAMŽITÉM ZÁPISU DAT

- **Fáze redo** – zopakování transakcí od kontrolního bodu (směrem vpřed), které nemají commit ani abort. Vytvoří se seznam undo (iniciálně L – aktivní transakce v momentě), přidá se tam transakce, pokud se narazí na <Ti start>, odstraní se odtud, pokud nalezne <Ti commit/abort>. Narazí-li na normální zázn. <Ti, Xj, V1, V2>, nahradí ho redo-only záznamem <Ti, Xj, V1>.
- **Fáze undo** – odčinění neukončených transakcí zpětným průchodem od posledního konktrolního bodu. Instrukce ze seznamu undo. Nalezne-li <Ti start>, zapíše <Ti abort> a Ti odstraní z undo.

## ŘÍZENÍ SOUBĚHU

- **Pesimistické plánovače**: předpokládají hodně konfliktů, předcházejí jim silnou serializací. Zamykací protokoly a čas. razítka.
- **Optimistické plánovače**: předpokládají málo konfliktů, málo blokují, nechávají transakce běžet. Konflikty se řeší rušením a opakováním konfliktních transakcí. Protokol na bázi validace serializovatelnosti plánu.

## OPERACE WRITE PŘI AUTOMATICKÉM GENEROVÁNÍ ZÁMKU

- Každá datová položka má zámek lock, k datové položce má přístup jenom transakce, která zamkla zámek.
- Zámky jsou sdílený pro read a exkluzivní pro write.
- Transakce jsou díky tomu synchronizované tak, aby jejich účinky na data byly ekvivalentní některému jejich sériovému zpracování.
- Transakce musí před přístupem získat od správce zámků (TPM) zámek položky a po použití ho správci vrací. Lock-S, lock-X, unlock.

## ČTENÁŘI-PÍSAŘI

- Čtenáři mohou k datové položce přistupovat hromadně, ale zápis (písaři) musí v danou chvíli provádět nejvýše jeden proces/transakce. Ze zapisované položky se nesmí číst.
- **Dva typy zámků**: sdílený (lock-S, slouží ke čtení) a exkluzivní (lock-X, slouží k současnému čtení a zápisu). Transakce musí před přístupem k datům Q vlastnit příslušný zámek (získat od správce zamků), po použití zámek uvolňuje.
- **Zprávy v procesu**: request(typ zámku, položka), lock-S(Q), lock-X(Q), unlock(Q).
- T žádá lock-X(Q) ⟶ pokud je Q zamčena (jakkoliv), musí T čekat
- T žádá lock-S(Q) a Q je zamčena exkluzivním zámkem ⟶ T musí čekat
- T žádá lock-S(Q) a Q je zamčena sdíleným zámkem ⟶ T získá přístup také

## PRODUCENT-KONZUMENT, ŘEŠENÍ SE SEMAFOREM

- Použití sdílené vyrovnávací paměti s omezenou kapacitou pro výměnu dat mezi dvěma procesy.
- Vždy před vybráním položky z bufferu se zvýší nebo sníží hodnota na semaforu atomickým příkazem.

## 2PLP

- Dvoufázový zamykací protokol. Zajišťuje konfliktovou serializovatelnost. Správa zámků má vyšší režii, neregulované zamykání může uváznout.
- Jakmile transakce odemkne zámek, nesmí už zamknout další.
- **Růstová fáze**: získávání zámků, serializování transakcí dle pravidel
  - pokud objekt není zamknutý, zamkne se a transakce pokračuje
  - pokud je zamknutý konflitkním zámkem jiné transakce, transakce čeká na odemknutí
  - pokud je zamknutý nekonfliktním zámkem a objekt se sdílí, transakce pokračuje
  - pokud je zamknutý stejnou transakcí, zamkne se a transakce pokračuje
- **Couvací fáze**: uvolňování zámků, jakmile se transakce provede nebo zruší, TPM odemkne všechny objekty transakce
- Pokud transakce uvolňují exkluzivní zámky až jsou ukončené, je to striktní protokol, pokud všechny, tak přísný protokol.
- Kaskádnímu vracení se dá vyhnout tak, že transakce, která čte nebo zapisuje objekt se musí pozastavit, než se jiná transakce, která zapisovala do téhož objektu, stane provedenou nebo zrušenou.

## ALGORITMUS VZÁJEMNÉHO VYLOUČENÍ V DISTR. PROSTŘEDÍ S FRONTOU A ČASOVÝMI RAZÍTKY

- Každá transakce má razítko dané dobou jejího vzniku. Při vzniku konfliktu přistupují transakce ke sdíleným proměnným v pořadí razítek. Pokud to nelze zajistit, transakce krachuje a restartuje se s novým razítkem.
- Jestliže transakce přepíše X, starší transakce už ji nemůže číst nebo přepsat. Když ji přečte, starší ji nemůže přepsat. Jestliže se nedaří najít pozitivní validaci časového pořadí konfliktní operace, transakce krachuje, ruší se rollback a restartuje se v novém čase.
- Požadavek write je valídní jen pokud byl objekt naposledy čtený/ zapisovaný starší transakcí nebo nebyl dosud přístupný.
- Požadavek read je validní jen pokud byl objekt naposledy čtený starší transakcí nebo nebyl modifikovaný.
- Časové razítko se odvozuje ze systémových hodin nebo z logického čítače inkrementovaného spouštěním transakcí.
- Wq – nejmladší transakce, která úspěšně provedla write(Q), Rq…
- Zajišťuje konfliktovou serializovatelnost, zamezuje uváznutí, nevylučuje možnost stárnutí.

## NA BÁZI VALIDACE

- Optimistický protokol, efektivní při malé pravděpodobnosti konfliktů. Hrozí stárnutí dlouhých transakcí. Brání kaskádám.
- Test validace se provádí při vydání požadavku na ukončení transakce. Cílem testu je zjistit, zda může transakce zapsat zpracovaná data, aniž by porušila serializovatelnost plánu souběžných transakcí.
- Transakce pracují s kopiemi objektů v RAM, neférové čtení nevzniká.
- Při ukončení transakce se testem validnosti ověří, zda došlo ke konfliktu. Pokud ano, některá z transakcí se zruší a zopakuje. Transakce bez zápisu se stane provedenou hned po validaci, se zápisy až po korekci DB. Nekonfliktní jsou také transakce dokončené před spuštěním validace a ty, které jsou ještě aktivní, ale jiné transakce ji nečtou/nepřepisují a ona je taky ne.

## CÍLE PLÁNOVAČE SOUBĚŽNOSTI TRANSAKCÍ S OHLEDEM NA PLÁN

- Zajištění serializovatelnost plánu za pochodu, garantování co nejvyšší propustnosti. Operace read a write se provádějí pouze v přípustném pořadí dle serializovatelného plánu bez ohledu na přidělování zdrojů.

## ŘEŠENÍ POMOCÍ MONITORU

- Monitory mají fronty procesů asociovaných proměnnou podmínky condition x,y; se dvěma operacemi x.wait(); - proces vyvolávající tuto operaci je potlačen až do doby, kdy jiný proces provede x.signal(); - operace aktivuje jeden proces z těch, co čekají na splnění x.

## MAJORITNÍ ZAMYKACÍ PROTOKOL

- Řeši decentralizované zamykání replikovaných dat. V každém uzlu běží správce zamykání uzlu, který má na starosti zamykání a odemykání dat/replik dat uchovávaných ve svém uzlu.
- Transakce zamykající replikovaný zdroj musí poslat požadavek na zamčení více než polovině uzlů, které uchovávají repliky zamykaných dat, získat od těchto uzlů povolení zamknutí, posléze požádat stejnou množinu uzlů o odemčení.
- Implementace → 2((n/2) + 1) zpráv pro lock (žádost, povolení), (n/2)+1 zpráva na odemčení unlock
- Algoritmus pro uváznutí musí být modifikován, protože uváznutí může způsobit i zamykání pouze jediné, ale replikované položky dat (Data v uzlech S1, S2, S3, S4, T1 žádá S1-3, dostane S1-2, T2 žádá S2-4, dostane S3-4 ‑ systém uvázl)

## GRAFOVÝ (STROMOVÝ) PROTOKOL ŘÍZENÍ SOUBĚHU TRANSAKCÍ

- – musí znát pořadí zpřístupňování datových položek → tvorba acyklického grafu
- – uzly - množina všech datových položek
- – di → dj = ve všech transakcích zpřistupňujících di i dj se di zpřístupňuje před dj
- – zajišťuje konfliktovou serializovatelnost (záměnou nekonfliktních operací lze získat sériové seřazení procesů, konfliktní operace = přístup ke stejné položce a aspoň jedna z nich píše)
- – lze použít pro řízení souběhu transakcí s exkluzivním zámkem lock-X
- – nedrží-li Ti žádný zámek, může zamčít libovolnou datovou položku
- – následně může Ti zamčít Q pouze tehdy, když už zamčela rodiče Q
- – zámek může transakce uvolnit kdykoliv
- – jedna transakce může zamknout-odemknout tutéž datovou položku pouze jedenkrát
- Pořadí zpřístupňování je definováno na základě logické či fyzické organizace dat, případně může být uměle vloženo pouze pro řízení souběžnosti apod.
- Výhody: zajištění konfliktové serializace, zabránění uváznutí, transakce se nemusí vracet, zvýšení souběžnosti (zámek lze uvolnit kdykoliv)
- Nedostatky: výsledný plán nezaručuje obnovitelnost, nelze zamezit kaskádnímu vracení, transakce musí formálně zamykat data, která nepotřebují (=> vyšší režie, potenciálně nižší souběžnost), pro danou akci mohou existovat konfliktově serializovatelné plány, které stromovým protokolem nelze získat
- Existují plány získatelné 2PLP (dvoufázový transakční protokol na bázi zamykání) a nezískatelné stromovým grafem a naopak.

# UVÁZNUTÍ

## NUTNÉ A DOSTAČUJÍCÍ PODMÍNKY UVÁZNUTÍ, ZPŮSOBY ŘEŠENÍ

- Existuje množina procesů, z nichž každý vlastní nějaký prostředek a čeká na prostředek vlastněný jiným z dané množiny.
- Podmínky uváznutí (pokud platí současně, 3 nutné, poslední postačující):
  – **vzájemné vyloučení** → sdílený zdroj může v jednom okamžiku používat pouze jeden proces
  – **požadavky se uplatňují postupně** → proces vlastnící aspoň jeden zdroj čeká na uvolnění zdroje vlastněného jiným
  – **nepřipouští se předbíhání** → zdroj lze uvolnit pouze procesem, který ho vlastní
  – **došlo k zacyklení požadavků** → existuje posloupnost n čekajících procesů, kde P1 čeká na P2, P2 na P3, … , PN na P1
- Metody zvládnutí uváznutí
  – **ochrana prevencí** → systém se do uváznutí nedostane, ruší se platnost některé nutné podmínky návrhovým rozhodnutím; přímá (zamezení nutné podmínky) x nepřímá (zamezení cyklům, uplné uspořádání typů zdrojů)
  – **obcházení uváznutí** → detekce potenciálního uváznutí, nepřipouští tento stav, zamezuje současné platnosti nutných podmínek rozhodnutími vydavanými za běhu, hrozí stárnutí; systém potřebuje dodatečné informace (nejjednodušši → maximum zdrojů), systém stanovuje bezpečný stav a bezpečnou posloupnost procesů (existuje nějaká posloupnost, kdy se systém nedostane mimo bezpečný stav); alogritmus RAG (nárokové, požadavkové hrany a hrany přidělení; graf procesů a zdrojů), bankéřův algoritmus
  – **detekce uváznutí a obnova po něm** → system detekuje uváznutí a provede definované akce; připouští se uváznutí, aplikuje se plán obnovy; algoritmus detekce (m x n x n operací pro detekci).

## STRATEGIE WAIT-DIE, A ZADA O ZDROJ B, C ZADA O ZDROJ B

- Wait-Die je jedním ze schémat řešení problému uváznutí při souběžných transakcích (druhým je Wound-Wait).
- Jestliže proces Pi žádá prostředek držený procesem Pj, porovná se jejich časová známka (TS). Je-li Pi starší než Pj (jeho časová známka je nižši), čeká (wait) na uvolnění prostředku, získá ho jakmile Pj prostředek uvolní. Je-li Pi mladší než Pj, je Pi vrácen zpět (umírá, dies) na vydání žádosti, Pi žádost zopakuje se stejnou časovou známkou.
- Starší procesy čekají na uvolnění, mladší se vrací a snaží se opakovaně vyhrát pozici staršího; čím je proces starší, tím více je mladších procesů, tím větší má šanci prostředek získat. Časem ho vždy získá, zabraňuje to stárnutí.
- Proces smí zemřít několikrát, ale v konečném čase prostředek získá (počká si na něj).
- Ad 1 ‑ A je starší, tj bude čekat do uvolnění prostředku
- Ad 2 ‑ C je mladší, takže bude vrácen na zopakování žádosti, časové razítko se mu zachová

## WOUND-WAIT, PRIKLAD TRANSAKCI, JAK SE BUDOU CHOVAT

- Stejně jako wait-die, jenom je prohozená role staršího a mladšího. Bývá tak méně návratů a více čekání.
- Transakce T1, T2 a T3 mají TS s hodnotami 5, 10, resp. 15
- Jestliže T1 (TS1 = 5) požaduje zámek držený transakcí T2 (TS2 = 10), přebere T1 zámek od T2, T2 je ,,zraněná" transakcí T1…
- Jestliže T3 (TS3 = 15) požaduje zámek držený transakcí T2 (TS2 = 10), transakce T3 bude čekat na jeho uvolnění.

## BANKÉŘŮV ALGORITMUS

- Myšlenka: Zákazníci si chtějí půjčovat prostředky z banky, předem deklarují maximální výši uvěrů, které v konečném čase splácí. Bankéř nemůže uvěr poskytnout, pokud si není jistý, že může uspokojit požadavky všech svých zákazníků.
- Algoritmus obchází uváznutí na základě předem definovaných maximálních požadavků zdrojů, užitečný při násobnosti instancí zdrojů. Proces požadující přidělení může čekat, ale díky tomu, že procesy v konečném čase zdroje uvolní, nebude čekat věčně. Vždy se předpokládá nejhorší případ (procesy využivající maxima naráz).
- Datové struktury: n → počet procesů, m → počet typů zdrojů
- – Available → vektor délky m, Available[j]=k = je dostupných k instanci zdroje Rj
- – Max → matice n x m, Max[i, j]=k = proces Pi bude používat nejvýše k instanci zdroje Rj
- – Allocation → matice nx m, Allocation[i,j]=k = Pi právě teď používá k instaci zdroje Rj
- – Need → matice n x m, Need[i,j]=k = Pi může pro dokončení požadovat ještě k instaci zdroje Rj, platí Need = Max -Allocation
- – Request[i,j]=k = Pi požaduje (teď) k instanci zdroje Rj

# KOORDINACE

## DME – PŘEDÁVÁNÍ PŘÍZNAKU, ČASOVÉ ZNÁMKY, SLOŽITOST

- DME = Distributed Mutual Exclusion, distribuované vzájemné vyloučení
- Předpoklady: systém se skládá z n procesů, který každý běží na jiném procesoru; zprávy vysílané jedním procesem jsou přijímané v pořadí jejich vysílání, doručené plně propojenou komunikační sítí (tj. bez mezilehlých procesů), doručí se v konečném čase; každý proces má kritickou sekci (KS), která se musí při běhu vzájemě vyloučit s ostatními sdruženými KS
- Požadavky/cíle: **podmínka bezpečnosti** (v jednom okamžiku smí KS provádět nejvýše jeden proces), podmínka **živosti** (jestliže proces žádá o vstup do KS, v konečném čase toto právo obdrží => nedojde k uváznutí ani ke stárnutí), podmínka **spravedlivosti** (žádající proces smí být každým jiným procesem distribuovaného systému předběhnut nejvýše jednou, pořadí vstupu do KS = pořadí podání žádosti)
- Plně distribuované řešení - **distribuovaná fronta**. Procesy se řadí do fronty podle času podání žádosti o vstup do KS. Každý proces si udržuje svůj **logický čas**, který odpovídá počtu žádostí o vstup do KS.
- – proces chce vstoupit do KS → inkrementuje TSi a pošle žádost všem procesům - request(Pi, TSi), čeká na odpověď
- – proces Pj, který obdrží žádost o vstup do KS, může odpovědět reply okamžitě nebo opožděně
- – je-li Pj v KS → reply pošle po opuštění KS
- – Pj nechce vstupovat do KS/nepožádal o vstup do KS → reply posílá okamžitě
- – Pj podal žádost o vstup do KS, ale dosud mu nebyl povolen → porovná svoji časovou známku (time stamp TS) TSj s TSi, je-li TSj > TSi (Pi žádal vstup do KS dříve) → reply posílá okamžitě, jinak nejdříve čeká na obdržení povolení o vstup do své KS od všech procesů, provede svoji KS, potom pošle reply
- – proces může vstoupit do KS po obdržení reply od všech procesů
- – po opuštěni KS posílá proces reply všem procesům, kterým dosud na žádost neodpověděl
- Pozitivní vlastnosti: splněny podmínky bezpečnosti, spravedlivosti i živosti (nedojde ke stárnutí, vyloučeno uváznutí, obsluha podle logického času ‑ first comes first served), minimální počet zpráv pro vstup do KS jednoho procesu je 2(n ‑ 1), kde n je počet procesů
- Negativní vlastnosti: proces musí znát identitu všech procesů v systému (netrivialní dynamické rušení a doplňování), výpadek jednoho procesu způsobí kolaps celého systému, protokol je vhodný pro malé a stabilní množiny kooperujících procesů.

## SUZUKI-KASAMI

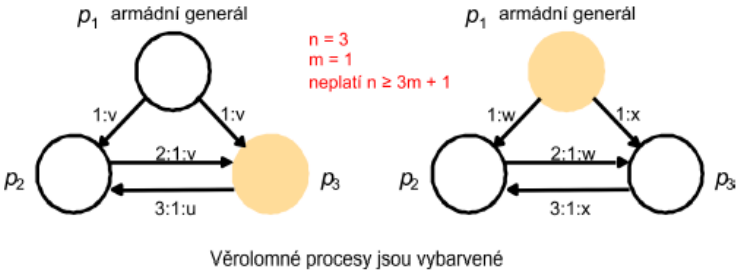
- Příznak povolení ke vstupu do KS na bázi běhu logického času v DS. Příznak je zpráva předávaná mezi procesy obsahující N prvkové pole T, ve kterém Tk udává čas posledního použití procesu Pk.
- Proces vstupující do KS všechny žádá zprávou request všechny procesy.
- Každý proces udržuje N prvkové pole R, ve kterém Ri udává časové razítko posledního request (Pi, TSi).
- Nejdřív drží příznak libovolný proces.
- Pokud chce Pi vstoupit a drží příznak, po vystoupení z KS provede test, komu má příznak předat (v příznaku vyhledává proces s Tk<Rk, hledání začíná od procesu, který naposled měl příznak). Pokud příznak nedrží, posílá request a čeká na příznak.
- Pokud Pi obdrží žádost Pj a drží příznak, provede test, pokud ho nedrží, registruje žádost v poli R.

## VOLEBNÍ ALGORITMUS PO KRUHU

- Volba probíhá, pokud vypadl koordinátor, je třeba zvolit nový časový server, ztratil se token a je třeba vygenerovat právě jeden nový apod. Důvod: programování v prostředí master/slave je jednodušši, ale dynamická volba je vhodnější pro méně stabilní prostředí.
- Volební algoritmus všeobecně:
- – nějaký proces vyvolá volbu, a to nejvýše jedenkrát → až n běhů volby zaráz
- – každý proces se buď volby učastní, nebo neúčastní
- – každý proces si udržuje informaci o zvoleném uzlu
- – bezpečnost → rozhodnutí uzlu být vedoucím procesem je procesem nezměnitelné, vedoucím procesem je běžící proces s nejvýšší prioritou, každý běžící proces má proměnnou elected buď prázdnou, nebo v ní má uloženou informaci o zvoleném uzlu (nejvyšší priorita)
- – podmínka živosti → každý uzel se nakonec dostane do stavu zvolený nebo podřízený (elect má neprázdnou), jeden z uzlů se stal vedoucím
- Každý proces má nadefinovanou cestu ke svému nasledovníku. Na začátku není žádný uzel učastníkem volby.
- Proces Pi vyhlašuje volbu, stává se učastníkem volby. Posíla election(Pi) svému sousedovi.
- Uzel Pj reaguje následovně (j = i+1):
- – Pi > Pj → Pj přepošle election(Pi)
- – Pi < Pj, Pj není učastníkem → pošle election(Pj), označí se za učastníka volby
- – Pi < Pj, Pj je učastníkem → zprávu neposílá
- – Pi = Pj → Pj získal zpět svoji zprávu, přepne se do stavu vedoucí uzel, označí se za neučastníka volby a pošle po kruhu zprávu elected(Pj). Uzel, který obdrží elected(Pj), si poznačí nový vedoucí uzel, označí se za neučastníka, a pokud není Pj tak zprávu pošle dál.
- Sledováni účasti na volbě pomáhá eliminaci duplicitních voleb. Složitost → nejvýše se pošle (3n-1) zpráv
- Varianta 2 → během volby si každý uzel vytváří seznam aktivních uzlů s jejich prioritami. Obíhá (n x n) zpráv. Po obnovení se vypadlý proces může po kruhu dotazovat na koordinátora.

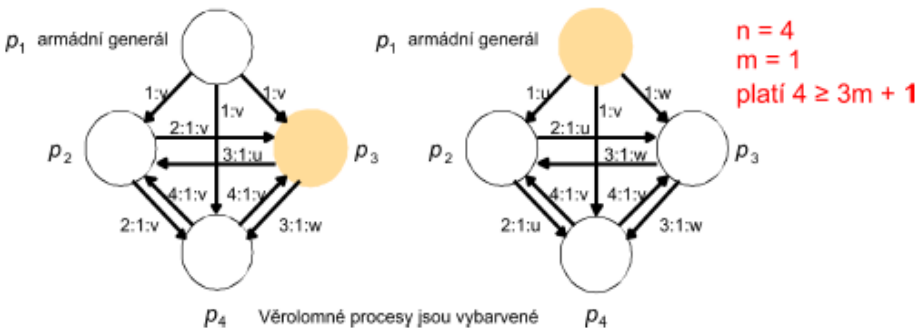
## BULLY ALGORITMUS (SOUPEŘENÍ KAŽDÝ S KAŽDÝM)

- Pokud uzel **zjistí výpadek koordinátora** (nedostane včas token/odpověď na zprávu), zkusí se prohlásit za nového koordinátora - **pošle všem zprávu** (zjišťuje, zda má nejvyšši prioritu) → musí všechny procesy znát (musí být možné, aby každý proces poslal zprávu všem ostatním procesům).
- Pokud **nedostane odmítnutí**, prohlašuje se za koordinátora. Pokud dostane odmítnutí od Pj, pak čeká na jeho prohlášení za koordinátora (Pj odstartoval volbu) → pokud ho nedostane, **startuje volbu znovu**.
- Každý proces, který není koordinátorem, může dostat zprávu o prohlášení koorinatorem od Pj → Pj je nový koordinátor (Pj ma vyšší prioritu), Pj startuje volbu (Pj má nižší prioritu, je třeba ho odmítnout a vyhlásit svoji volbu).
- Proces po výpadku startuje stejný algoritmus → dozví se, kde je šéf (nebo je zvolen on).
- Složitost **O(n x n)** v nejhorším případě, tj. Když volbu zahájí proces s nejnižši prioritou.




## PROBLÉM BYZANSTKÝCH GENERÁLŮ

- Problém dosažení shody v distribuovaném prostředí.
- Předpokladem je, že generálové mohou havarovat a chovat se zákeřně – slyší A, říká B. Skupina generálů rozesílá zprávy, armádní posílá rozkaz divizním generálům a ti se rozhodují na základě toho, co řekl on a ostatní divizní, věrolomný generál řekne jednomu útočit a jinému ustupovat. Každý se dozví návrh od všech a nakonec rozhodne majoritní názor
- Podmínky možnosti shody: 1 z 5, 2 z 7, 5 z 16



# STRÁNKOVÁNÍ

## STRÁNKOVÁNÍ S TLB

- MMU: Počítačová komponenta zodpovědná za zpracování přístupů k paměti, o které žádá CPU (Central Processing Unit). Překládá virtuální adresu na fyzickou adresu, dále obsahuje nástroje pro ochranu dat, kontrolu cache a další.
- **TLB** (Translation Look-aside Buffer) je asociativní pamět', do které jsou ukládány dvojice (klíč,hodnota), kde klíčem je číslo logické stránky a hodnotou je odpovídající položka tabulky stránek. Slouží ke zrychlení práce MMU.

## SEGMENTACE A STRÁNKOVÁNÍ (SPOLEČNÉ RYSY A ROZDÍLY)

- **Segmentace**: LAP tvořen několika segmenty (oblastmi paměti obecně různé velikosti do limitu daného rozsahem adres (offsetu) v segmentu). Program při tomto způsobu musí obsahovat informace, ve kterém segmentu leží požadovaná data nebo kód. Náročnější na programátora než stránkování na žádost, může způsobovat fragmentaci fyzické paměti, prakticky se nepoužívá.
- **Stránkování na žádost:** Provádí mapování po stránkách (nejmenší jednotka přidělování paměti procesu a její velikost musí být mocninou 2). Adresový prostor si lze představit jako **pole stránek**, kde **indexy odpovídají číslu stránky**.
- LAP i FAP jsou rozděleny na stejně velké stránky. Stránce ve FAP obvykle říkáme rámec (frame). Stránka je **nejmenší jednotkou,** pro kterou lze nastavit ochranu paměti (právo číst, zapisovat, provádět kód) a sdílení paměti mezi procesy.
- Důležitou vlastností tohoto systému virtuální paměti je to, že proces se nemusí starat o mapování LAP na FAP, protože toto mapování je transparentně (tj. pro proces neviditelně) zajišťováno kombinací technického vybavení (**hardware**) a **jádra** OS.
- Implementačně složité, vyžaduje vyšší režii, v současnosti to používají všechny moderní víceúlohové systémy.

# DETEKCE UVÁZNUTÍ

## WAIT-FOR GRAFY S EXTERNÍM STAVEM REPREZENTOVANÝM UZLEM PEX

- wait-for graf = graf, jehož uzly jsou procesy, orientované hrany reprezentují čekání na zdroj držený jiným uzlem; P1 → P2 = P2 drží zdroj, na který P1 čeká
- Lokalni wait-for graf - uzly odpovídají lokálním procesům a procesům, které nejsou lokální, ale požadují zdroje lokální v daném uzlu distribuovaného systému. Globální wait-for graf - sjednocení lokálních wait-for grafů
- Existence uváznutí → je-li cyklus v lokálním nebo globálním grafu (problém: zpoždění)

## POMOCÍ PEX (PLNĚ DISTRIBUOVANÉ ŘEŠENÍ)

- – za detekci uváznutí sdílejí odpovědnost všichni koordinátoři
- – každý uzel sítě konstruuje wait-for graf, který reprezentuje část globálního wait-for grafu
- – každý lokální wait-for graf obsahuje uzel Pex (Proces Externí), platí: 1) Pi → Pex = uzel Pi čeká na data držená procesem ve kterémkoliv jiném uzlu sítě; 2) Pex → Pi = proces v libovolném jiném uzlu sítě čeká na data držená Pi
- – pokud lokální wait-for graf obsahuje cyklus neobsahující Pex, je ve stavu uváznutí
- – pokud lokální wait-for graf obsahuje cyklus, ve kterém je Pex, existuje možnost uváznutí, je nutné spustit distribuovaný algoritmus detekce uváznutí
- – lokální síť S1 odhalí cyklus obsahující Pex (P3 → Pex, Pex → P2), zjistí, že P3 žádá zdroj z uzlu sítě S2, pošle S2 zprávu popisující zjištěný cyklus, S2 upraví svůj lokální graf za pomocí informací z S1 a odhalí uváznutí

# DALŠÍ

## OPERAČNÍ SYSTÉMY S MIKROJÁDREM A OPERAČNÍ SYSTÉMY S VRSTVENÝM (MONOLITICKÝM) JÁDREM

- **Mikrojádro** provádí většinu abstrakcí práce s pamětí, správu přerušení, práci s procesy a vlákny, zajišťuje meziprocesovou komunikaci. Služby OS jsou poskytovány jako procesy nad mikrojádrem, lze emulovat více OS souběžně. Mezi procesy se komunikuje předáváním zpráv.
- **Monolitické jádro** provádí v jednom celku všechny služby poskytované OS, je obtížně manipulovatelný, minimální modulovatelnost, vysoká provázanost funkcí, služby jádra jsou typicky řešeny sekvenčně.

## ZÁKLADNÍ VLASTNOSTI VALIDNÍCH DISTRIBUOVANÝCH ALGORITMŮ

- **Bezpečnost** (Nothing bad happened yet) - Globální stav distribuovaného systému (DS) je ve stavu, ze kterého normálními stavovými přechody není dosažitelný jiný, nežádoucí stav (uváznutí, násobny vstup procesů do kritické sekce). Bezpečnost se typicky dokazuje indukcí, narušení bezpečnosti se prokazuje v konečném počtu kroků, řešení problému nenarušující bezpečnost je konkretní řešení.
- **Živost** (Something good eventually happens)- Vlastnost zajišťující, že jistou posloupností normálních stavových přechodů je dosažitelný jistý konkrétní žádoucí stav (zvolí se vedoucí uzel, proces žádající o vstup do KS toto právo obdrží). Narušení podmínky se dokazuje v nekonečném počtu kroků. Korektní řešení problému nenarušující živost je uplné, kompletní řešení.

## 2PCP A CO SE STANE, KDYŽ VYPADNE KOORDINÁTOR

- 2PCP – synchronizační protokol řešící problém atomicity.
- Distribuovaná transakce - uzly, které se na řešení transakce podílejí (participanti) tvoří tzv. kohortu; koordinátor transakce = řidí start a konec distribuované transakce v uzlech; participanti se registrují u koordinátora pomocí join zprávy, společně kooperují na řešení commit protokolu. 2PCP umožňuje každému participantovi jednostranně ukončit (abort) svoji část transakce → krachuje celá transakce (vlastnost atomicita)
- Koordinátor koordinuje ukončení transakce T jedním ze dvou způsobů
- – commit → pokud jsou portvzeny všechny subtransakce, transakce se převede do stavu provedena
- – abort → zruší účinky všech subtranskací, pokud alespoň jedna z nich zkrachovala, zrušení celé transakce
- **Faze 1 ‑ Can commit?** Každý participant hlasuje, zda může ukončit (commit) nebo ne (abort); pokud hlasuje pro ukončení, musí si být jistý, že bude schopen svoji část dokončit, i když mezitím vypadne a obnoví svoji činnost → všechny změněné objekty i status připravený k ukončení musí mít poznamenány v trvalé paměti
- **Faze 2 ‑ Do commit/Do abort** Pokud žádný participant nehlasoval abort, koordinátor pošle Do commit, v opačném případě pošle zprávu ke zrušení transakce Do abort
- **Problémy**: 2PCP vyžaduje hlasování všech participantů; musí správně pracovat i v prostředí vypadků serverů, ztrát zpráv, dočasného výpadku komunikace mezi servery. 2PCP dosahuje shody za omezující podmínky → výpadky procesů/serverů jsou maskovány obnovou vypadlých procesů novými procesy, jejichž stav je nastavený podle informací uchovávaných v permanentní paměti a informací držených jinými procesy; proces možná po nějakou dobu nereaguje, ale pokud reaguje, reaguje validně; podpůrný komunikační systém odstraní duplikované či poškozené zprávy
- Každý uzel si musí uchovávat deník (log), aby bylo možné transakci zrušit a/nebo provést znovu (undo, redo)
- Stav neurčitosti ‑ participant ve fázi 1 hlasoval commit a čeká na sdělení rozhodnutí, nemůže uvolnit prostředky, nemůže rozhodnout jednostranně.
- Řešitelnost dohody se řeší pomocí časových vypadků → time-out nemusí implikovat trvalou poruchu
- **Výpadek koordinátora:** Participant ve stavu neurčitosti musí čekat na obnovení činnosti koordinátora, po uplynutí časového limitu může požádat o zopakování zprávy o rozhodnutí, ale po x takových opakování krachuje. Participant se také může zeptat ostatních, pokud jsou tito také ve stavu neurčitosti, nedozví se nic. Participant, který ukončil svoji část transakce, ale ve stanoveném limitu nedostal dotaz Can commit → jednostranně ukončí transakci abort.
- **Výpadek participanta:** Koordinátor v časovém limitu neobdržel hlas od nějakého participanta → abort všem participantům. Pokud nějaký proces hlasuje commit se zpožděním, zůstane ve stavu neurčitosti. Uzel po obnově rozhoduje podle deníku (log) o osudu rozpracovaných transakcí: <commit T> → redo(T) (rozhodnutí bylo commit), <abort T> → undo (T)

## ROUND-ROBIN

- Round Robin, máme procesy P1, P2, P3, P4 zařaděné do fronty v čase 0, časy potřebné na dokončení procesů P1..P4 jsou 53, 17, 68, 24. V jakém pořadí skončí? Skončí v pořadí P2, P4, P1, P3, round-robin přiděluje jisté stále časové úseky pravidelně po kruhu.