Termín 5. 1. 2010

Popište algoritmus wound-wait. Pokud máme tři procesy s časovým razítkem 1 (A), 10 (B) a 20 (C), zámek X drží B, jak se zachovají ostatní procesy? - *Transakce a souběžnost, str. 53*

Starší proces zdroj odebere, původní vlastník se vrací zpět (undo). Mladší čeká na uvolnění zdroje starším procesem.

Popište volební algoritmus probíhající po kruhu se seznamem aktivních procesů. Určete složitost. - *Koordinace a dohoda, str. 44*

2PCP + zachování uzlů při výpadku koordinátora - *Transakce a souběžnost, str. 11*

DME – předávání příznaku, časové známky. Určete složitost. - *Koordinace a dohoda, str. 14*

Obnovení procesů na základě kontrolních bodů. - *Recovery, str. 27*

Co je to serializovatelný plán? Co znamená obnovitelný serializovatelný plán? Jak lze zabránit kaskádovému vracení obnovitelného časového plánu?

Prosinec 2009 – přibližné zadání. U dvou byl požadavek charakterizovat složitost.

Wait-for grafy s externím stavem reprezentovaným uzlem Pex - *Transakce a souběžnost, str. 61*

„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

Lokální „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 grafu, nebo je-li cyklus v 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 = 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
- S1 odhalí cyklus obsahující Pex (P3 → Pex, Pex → P2), zjistí, že P3 žádá zdroj z uzlu S2, pošle S2 zprávu popisující zjištěný cyklus, S2 upraví svůj lokální graf za pomocí informací z S1 a odhalí uváznutí

DME plně distribuované řešení - *Koordinace a dohoda, str. 20*

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 kritickými sekcemi

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 žádostí)

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 opuštění 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 (netriviální dynamické rušení a doplňování), výpadek jednoho procesu způsobí kolaps celého systému, protokol je vhodný pro malé a stabilní množiny kooperujících procesů

Grafový (stromový) protokol řízení souběhu transakcí - *Concurrency, str. 21*

- musí znát pořadí zpřístupňování datových položek → tvorba acyklického grafu
  - uzly – množina všech datových položek
  - di → dj = ve všech transakcích zpřístupňujících 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

2PCP (Two-phase Commit protocol) + řešení výpadku uzlu - *Transakce a souběžnost, str. 11*
Distribuovaná transakce – uzly, které se na řešení transakce podílejí (**participanti**) tvoří tzv. kohortu; **koordinátor transakce** = řídí 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 provedená

- abort → zruší účinky všech subtranskací, pokud alespoň jedna z nich zkrachovala, zrušení celé transakce

Fáze 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 poznamenaný v trvalé paměti
Fáze 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 výpadků 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 odstaní 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 výpadků → 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 participantů, 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
Když uzel s koordinátorem v rozpracovaném 2PCP vypadne, o osudu T rozhodují uzly kohorty. Pokud to nezvládají, čekají na obnovení činnosti koordinátora.
Pokud aktivní uzel s koordinátorem ve svém logu obsahuje <commit T>, T se potvrzuje
pokud obsahuje <abort T>, T se ruší
Pokud některý aktivní uzel ve svém logu neobsahuje <ready T>, nemohl koordinátor rozhodnout o provedení commit T a daný aktivní uzel tedy nemohl svoji část T potvrdit → nečeká se na obnovu koordinátora, T v tomto uzlu se ruší
Pokud aktivní uzel ve svém logu obsahuje <ready T>, ale nikoliv rozhodnutí commit/abort, musí čekat na obnovu koordinátora => T a všechny odpovídající zdroje držené v uzlech kohorty jsou do obnovy koordinátora blokované

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)
<ready T> → dotazuje se koordinátora (akce dle odpovědi, commit → redo, abort → undo); pokud koordinátor neodpovídá, uzel vysílá periodicky dotaz koordinátorovi a ostatním uzlům kohorty query-status(T), dokud nedostane informaci, zda byla transakce ukončená nebo zrušená a provede příslušné akce

log neobsahuje žádný ukončovací záznam o T → uzel vyšle abort, koordinátor musí transakci zrušit, uzel kohorty provede bezodkladně undo(T)

Majoritní zamykací protokol - *Transakce a souběžnost, str. 40*
Řeší 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)

Bankéřův algoritmus - *Uváznutí, str. 26*
Myšlenka: Zákazníci si chtějí půjčovat prostředky z banky, předem deklarují maximální výši úvěrů, které v konečném čase splácí. Bankéř nemůže úvěr poskytnout, pokud si není jistý, že může uspokojit požadavky všech svých zákazníků alespoň v jedom pořadí uspokojení.
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 využívající maxima naráz)
Datové struktury

- n → počet procesů
- m → počet typů zdrojů
- Available → vektor délky m, Available[j]=k = je dostupných k instancí zdroje Rj
- Max → matice n x m, Max[i, j]=k = proces Pi bude používat nejvýše k instancí zdroje Rj
- Allocation → matice nx m, Allocation[i,j]=k = Pi právě teď používá k instací zdroje Rj
- Need → matice n x m, Need[i,j]=k = Pi může pro dokončení požadovat ještě k instací zdroje Rj, platí Need = Max – Allocation
- Request[i,j]=k = Pi požaduje (teď) k instancí zdroje Rj

Algoritmus testu stavů
Používá pomocné vektory Work (délky m) a Finish (délky n), inicializace Work = Allocation, Finish[i]=false pro všechna i od 1 do n. Simuluje ukončení pro procesy, které mohou být ukončeny se zdroji ve Work, pokud ano, Finish[i] = true, Work = Work + Allocation[i]. Na konci testuje, zda jsou všechny procesy Finish = true, pokud ano, je systém v bezpečném stavu, pokud ne, není.
Algoritmus vyžádání zdroje
Test, zda nepřekročil deklarované maximum → případný konec chybou
Pokud zdroj není dostupný (Available), musí čekat. V opačném případě se provede test přidělení → pokud po přidělení je systém ve stavu bezpečný, přidělení se provede, jinak musí proces čekat
Běhy procesů z bezpečného stavu nezpůsobí uváznutí, stav, který není bezpečný, ještě neimplikuje uváznutí (procesy nemusí využívat deklarovaná maxima zdrojů). Předpokládá, že procesy se synchronizačně neomezují. Obcházení zamezí uváznutí, ale ne stárnutí.

Termín 14. 1. 2009
Jak funguje algoritmus vzájemného vyloučení v distribuovaném prostředí s pomocí distribuované fronty a časových razítek/předávání si zpráv? - *Koordinace a dohoda, str. 20*
Vysvětlete princip použití kontrolních bodů (checkpoint) při zpracování (obnově?) transakcí/ v implementaci transakcí. - *Recovery, str. 27*
Každá transakce si udržuje deník (log) se zápisem transakcí pro undo a redo akce. Tyto deníky

mohou být dlouhé, a při obnově může docházet ke zbytečnému a zdlouhavému obnovování dat, které již mají provedené zápisy (output) do báze dat → cílem kontrolních bodů je zkrátit dobu obnovy
Kontrolní body se vytvářejí periodicky, během tvorby nesmí transakce provádět žádnou akci typu korekce dat, zápis dat. Provádí se: výstup všech deníkových záznamů z energeticky závislé paměti (RAM) do stabilní paměti, výstup všech modifikovaných bloků z vyrovnávací do stabilní paměti, výstup záznamu <checkpoint> do logu ve stabilní paměti.
Transakce provedené před zápisem <checkpoint> se neobnovují pomocí redo (jejich výsledek je součástí dat kontrolního bodu); obnova se týká pouze akcí, které nebyly provedeny (commit) před vytvářením kontrolního bodu a všech transakcí započatých po tvorbě kontrolního bodu. Mají-li transakce v deníku <commit>, provede se redo (opakování), jinak se provede undo (eliminace).
Příklad:
T1 začala, T2 začala, T1 skončila, T3 začala
<checkpoint>
T4 začala, T4 skončila, T2 skončila, T5 začala
chyba
výsledek T1 je obsažen v kontrolním bodu → žádné akce
T2, T4 commit → redo v pořadí T4, T2 (hledá se konec od checkpointu)
T3, T5 nejsou provedené → undo v pořadí T5, T3 (hledá se začátek od chyby)

ACID - zakladni vlastnosti transakci - *Transkace, str. 6*
A = Atomicity (atomicita, all or nothing) → buď se provedou všechny operace dané transakce, nebo ani jedna. Transakce buď skončí úspěšně (výsledky všech operací se zaznamenají v relevantních objektech), nebo selže/je zrušená (transakce nevykáže žádný účinek). Systém musí zajistit, aby částečně provedené transakce neovlivnily stav báze dat.
Idea zajištění: systém podporující transakční zpracování si uchovává původní hodnoty dat, nad kterými operace příslušné transakce pracují. V případě nedokončení transakce je schopen obnovit původní stav báze dat (jako by transakce nikdy nezačala). Po úspěšném provedení transakce se výsledky zapíší do permanentní paměti (taková data „přežijí – D = Durability)

C = Consistency (konzistence báze dat) → musí se zajistit, že provedením transakce se nenaruší konzistence báze dat, ověřit lze např. Validním integritním omezením (A+B=const); nově spuštěná transakce musí vždy vidět validní bázi dat; chybně provedená transakce by mohla porušit validitu; během provádění transakce může být validita dočasně porušena

I = Isolation (izolovanost transakce) → souběžné provádění více transakcí nesmí ovlivňovat výsledek jednotlivých transakcí; systém musí zajistit, aby pro každý pár souběžných transakcí Ti, Tj se z hlediska každé transakce souběh jevil jako by buď Ti začala po konci Tj, nebo jako by Tj začala po konci Ti; triviálně lze zajistit plnou serializací (neefektivní), nebo řízením souběžnosti

D = Durability (trvalost výsledků transakcí) → jakmile se transakce úspěšně ukončí, změny jí provedené jsou trvalé bez ohledu na možný budoucí výpadek systému. Idea: změny provedené T se před ukončením zapíší na disk, a při obnově systému po poruše se tyto informace použijí pro rekonstrukci změn.

Uvedte a popiste 2 vlastnosti validnich distribuovanych algoritmu.
Bezpečnost (Nothing bad happened yet)
Globální stav distribuovaného systému (DS) je ve stavu, ze kterého normálními stavovými přechody není dosažitelný jiný, nežádoucí stav (uváznutí, násobný vstup procesů do kritické sekce). Bezpečnost se typicky dokazuje indukcí, narušení bezpečnosti se prokazuje v konečném počtu kroků, řešení problému nenarušující bezpečnost je konkrétní řešení.

Živost (Something good eventually happens)
Vlastnost zajišťujicí, 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). Narušen podmínky se dokazuje v nekonečném počtu kroků. Korektní řešení problému nenarušující živost je úplné, kompletní řešení.

Vysvetlete princip Wait-Die. Mame tri transakce kazda s urcitym casovym razitkem (A 17, B 21, C 26). Co se stane kdyz bude transakce A zadat o zdroj, drzeny transakce B? Co se stane kdyz bude transakce C zadat o zdroj, drzeny transakce B? - *Transakce a souběžnost, str. 51*

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 nižší), čeká (wait) na uvolnění prostředku, získá ho jakmile Pj prostředek uvolní. Je-li Pj 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.
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

Termín: 7. 1. 2009

Definovat problem čtenáři x písaři a řešení se semaforem s prioritou čtenářů- *??*
Concurrency, str. 5 - asi

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 správe zámků), 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 zamčená (jakkoliv), musí T čekat
T žádá lock-S(Q) a Q je zamčená exkluzivím zámkem → T musí čekat
T žádá lock-S(Q) a Q je zamčená sdíleným zámkem → T získá přístup také

Paměť má velikost 1GB. Máme procesy A,B,C,D, které mají přiděleno 100MB, 100MB, 200MB a 400MB. Maximálně mohou žádat 400MB, 500MB, 500MB, 600MB. Správa paměti se dělá pomocí bankéřova algoritmu. Proces B žádá o přidělení 100MB. Bude jeho požadavku vyhověno? Zdůvodněte! - *Uváznutí, str. 26*

Bakalářův algoritmus – viz výše.

Původní stav (Available = 200)

	Allocated	Need	Max
A	100	300	400
B	100	400	500
C	200	300	500
D	400	200	600

Předstírané přidělení (Available = 100)

	Allocated	Need	Max
A	100	300	400
B	200	400	500
C	200	300	500
D	400	200	600

Systém může předstírat přidělení, protože requested <= Need, Requested <= Available
Test systému, zda je možné uspokojit procesy => nelze (všechny procesy mají na konci Finished = false) → požadavku nebude vyhověno

Znázorněte překlad logické adresy na fyzickou pomocí invertované tabulky. Když má logická paměť kapacitu 1GB a stránka má délku 1kB, velikost FAP je 1 MB, kolik řádků bude mít

invertovaná tabulka? - *??*
Uváznutí - definice + všechny 4 podmínky + popsat 3 metody řešení - *Uváznutí, str. 2*
Existuje množina procesů, z nichž každý vlastní nějaký prostředek a čeká na prostředek vlastněný jiným z dané množiny procesů.
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 zdrije vlastněného jiným procesem
- 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, úplné uspořádání typů zdrojů)
- obcházení uváznutí → detekce potenciálního uváznutí, nepřipouští tento stav, zamezuje současné platnosti nutných podmínek rozhodnutími vydávanými za běhu, hrozí stárnutí; systém potřebuje dodatečné informace (nejjednodušší → 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 → systém detekuje uváznutí a provede definované akce; připouští se uváznutí, aplikuje se plán obnovy; algoritmus detekce (m x n x n operací pro detekci)

Co je to serializovatelný plán. Kolik je sériových plánů při realizaci 4 transakcí?
Sériový plán = instrukce náležející jedné transakci tvoří kontinuální blok
Plán = vymezuje možná pořadí provádění instrukcí transakcí
Serializovatelný plán = je-li ekvivalentní některému sériovému plánu stejné skupiny transakcí
Idea zajištění serializovatelnosti – konfliktová serializace
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 operací.
Sériových plánů při realizaci 4 transakcí je 4! → na pořadí transakcí nezáleží.

Písemka 09.01. 2008
Nutné podmínky pro uváznutí. - *Uváznutí, str. 2*
Paměť má velikost 1GB. Máme procesy A,B,C,D, které mají přiděleno 100MB, 100MB, 200MB a 400MB. Maximálně mohou žádat 400MB, 500MB, 500MB, 600MB. Správa paměti se dělá pomocí bankéřova algoritmu. Proces B žádá o přidělení 100MB. Bude jeho požadavku vyhověno? Zdůvodněte! - *Uváznutí, str. 26*
Popište vlastnosti transakce – ACID. - *Transakce, str. 6*
Vysvětlete prevenci uváznutí transakcí pomocí Wait-Die. - *Transakce a souběžnost, str. 51*

Termín: 24.1.2007
Tranzakcie + ACID - *Transakce, str. 6*
uviaznutie - definicia + vsetky 4 podmienky + 3 sposoby riesenia - *Uváznutí, str. 2*
nejaka volba koordinatora medzi procesmi pomocou sprav ELECT, kolko sprav sa odosle- *??*
asi to má být Ring Algorithm, varianta 2
Koordinace a dohoda, str. 27 – asi...
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

jednodušší, 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 účastní, nebo nepúčastní
- výsledek volebního procesu musí být jednoznačný bez ohledu na počet probíhajících voleb
- každý proces si udržuje informaci o zvoleném uzlu
- bezpečnost → rozhodnutí uzlu být vedoucím procesem je procesem nezměnitelné, vedoucím procesem je běžící proces s nejvyšší 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

Bully algorithm (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 nejvyšší 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 koorinátorem od Pj → Pj je nový koordinátor (Pj má 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 nejnižší prioritou

Ring algorithm (volba směrovaná po komunikačním kruhu)
Každý proces má nadefinovanou cestu ke svému následovníku. Na začátku není žádný uzel účastníkem volby.
Proces Pi vyhlašuje volbu, stává se účastníkem volby. Posílá election(Pi) svému sousedovi.
Uzel Pj reaguje následovně (j = i+1):

- Pi > Pj → Pj přepošle election(Pi)
- Pi < Pj, Pj není účastníkem → pošle election(Pj), označí se za účastníka volby
- Pi < Pj, Pj je účastníkem → zprávu neposílá
- Pi = Pj → Pj získal zpět svoji zprávu, přepne se do stavu vedoucí uzel, označí se za neúčastníka volby a pošle po kruhu zprávu elected(Pj)

Uzel, který obdrží elected(Pj), si poznačí nový vedoucí uzel, označí se za neúčastníka, a pokud není Pj tak zprávu pošle dál
Sledování 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.
producent - konzument, riesenie pomocou semaforu - *??*

PA150 ZK 20. 12. 2006
Round Robin, mame procesy P1, P2, P3, P4. zaradene do fronty v case 0, casy potrebne na dokoncenie procesov P1..P4 su 53, 17, 68, 24. V akom poradi skoncia? - *??*
Definujte problem producent-konzument, riesenie so semaforom. - *??*
Strankovanie a segmentacia - vyznam, spolocne a rozdielne vlastnosti. - *??*
Deadlock - definicia, 4 podmienky, 3 sposoby riesenia. - *Uváznutí, str. 2*
3 procesy P1-P3, P3 zahajuje volbu koordinatora, algoritmus elect, kolko sprav elect(i) bude odoslanych, ked alg. skonci v kazdom P - *??*

Def. tranzakcie, vlastnosti ACID - *Transkace, str. 6*
Distribuovany system se 3 pocitaci v kruhu a kolik zprav se musi poslat pro volbu noveho koordinatora a nakresli topologii chodu zprav
Koordinace a dohoda, str. 27
Nejhorší případ, P1<P2<P3, P1 zahájí volbu, pošle election(P1) → P2 pošle election(P2) → P3 pošle election(P3), kterou P1 i P2 přepošlou, P3 pošle elected(P3), ta je také přeposlaná. Počet zpráv: 3 (vytvořené election) + (3-1) (přeposlané election) + 3 (elected) (n + n -1 + n = 3n – 1)

21.12.2005
Round Robin - *??*
Prioritni planovani s predbihanim, bez predbihani - *??*
S předbíháním – Round Robin (každý proces má přidělenou časovou jednotku, pak je zařazen opět na konec fronty
Bez předbíhání – procesy jsou zařazeny do prioritních tříd, procesy ve stejné třídě se střídají Round Robin, procesy s nízkou prioritou hladoví a stárnou
Producent x konzument + reseni pomoci semaforu - *??*
Uvaznuti - nutne podminky, reseni uvaznuti, definice - *Uváznutí, str. 2*
Segmentace a strankovani - spoledne rysy, rozdily - *??*
Transakce + ACID - *Transkace, str. 6*
LRU algoritmus + 2 aproximace - *??*

12.1.2005
LAP a FAP - *??*
3 vlastnosti KS, popiste Petersonuv algoritmus reseni KS, kterou podminku nesplnuje

3 vlastnosti planovani FSS (spravedlnost, efektivita, pruchodnost), vypocet i doby prideleni CPU v zavislosti na skupine procesu k

EAT - ve virtualizaci se strankami v operacni pameti a s moznosti zapisu stranek - *??*
Disk 10GB, bitova mapa kolik procent disku zabira, kdyz alokacni blok ma velikost 4KB - *??*
V distribuovanych systemech vlastosti majoritniho zamykaciho protokolu a pocet zaslanych zprav pri odemykani a zamykani - *Transakce a souběžnost, str. 40*

PA150 Principy operacnich systemu, Doc. Jan staudek, 28/01/04
Co je to DMA? - *??*
Ilustrujte funkci monitoru na typove synchronizacni uloze - *??*
Definujte uvaznuti. Napiste nutne podminky a postacujici podminku uvaznuti. Napiste vsechna tri pouzivana reseni uvaznuti - *Uváznutí, str. 2*
Odvodte a ukazte na prikladu co je to EAT(Effective Access Time). - *??*
Popiste exponencialni prumerovani pro planovani cinnosti procesoru. - *??*
Charakterizujte operacni systemy s mikrojadrem a operacni systemy s vrstvenym (monolitickym) jadrem.
Monolitické jádro
Provádí v jednom celku všechny služby poskytované OS, je obtížně manipulovatelný, minimální modulovatelnost, vysoká provázanost funkcí, služby jádra jsou typicky řešeny sekvenčně
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.