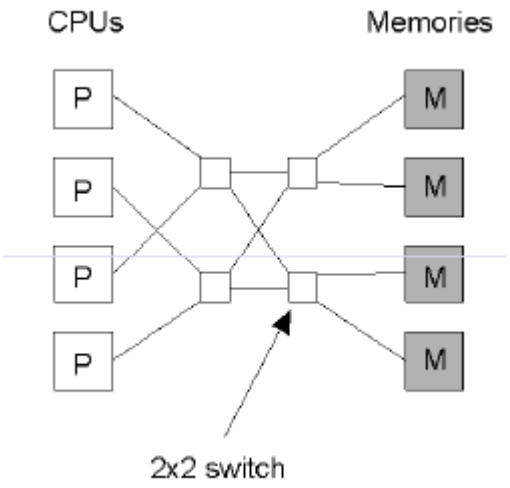
Příklady písemných prací ke zkoušce z DS

Distribuovaný systém = soubor nezávislých počítačů, které se jeví svým uživatelům jako jeden souvislý systém. Účel: zvýšení výkonnosti, zvýšení dostupnosti, zvýšení spolehlivosti. Modely: klient/server, p2p.

1. Načrtněte architekturu multiprocesorového systému se čtyřmi procesory a čtyřmi pamětmi s omega architekturou.

Multistage switched (omega networks) (propojeni pamětí a procesorů). Pro tento případ nám stačí 2 bity pro propojení jakéhokoliv procesoru s jakoukoliv pamětí.

1. Postupně přepínaná cesta mezi procesorem a pamětí
2. Při větším počtu stupňů pomalé
3. $n*log(n)$ přepínačů

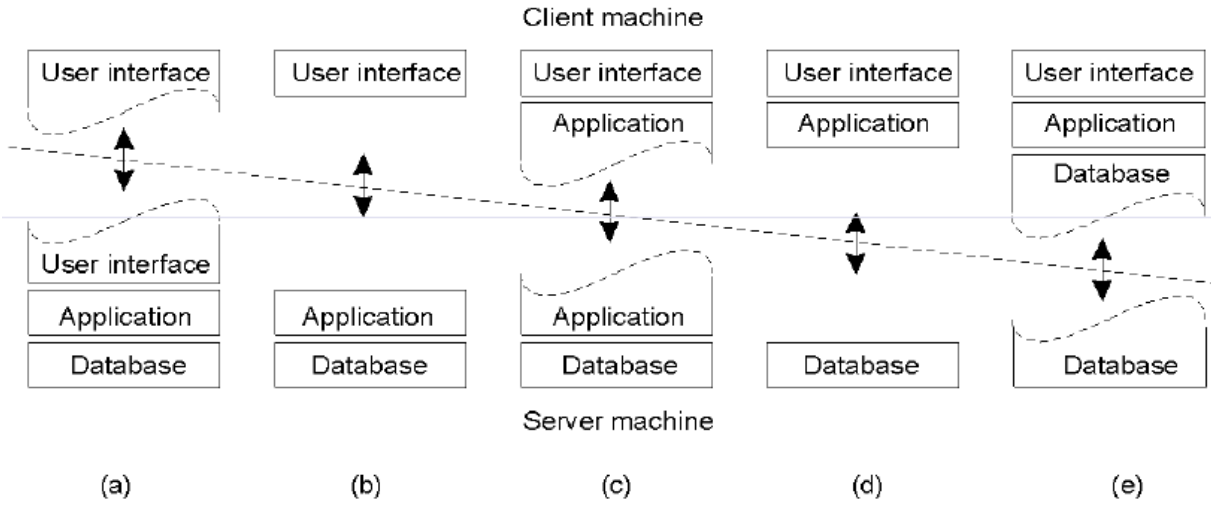


2. Zakreslete obrázek s možnými typy třívrstvé architektury klient/server (uživatelské rozhraní, aplikace a databáze), uveďte příklady použití a zhodnoťte je.

Prezentační vrstva – obsahuje funkce uživatelského rozhraní. Obvykle existuje několik prezentačních vrstev pro různé druhy zařízení, platformy a prostředí

Aplikační vrstva – tvoří prostředníka mezi vrstvou prezentační a vrstvou datovou. Obsahuje tzv. business logiku aplikace. V této vrstvě dochází k transformaci dat mezi vstupně / výstupními požadavky a datovou vrstvou.

Datová vrstva – obsahuje funkce pro přístup k informacím v datovům úložišti



(a) Přes internetové prohlížeče se přenáší data na server, který je pak zpracuje (např. javascript zkontroluje, zda byly vyplněny všechny údaje)

(b) Konzole na klientovi, interpret příkazů a databáze na serveru

(c) …

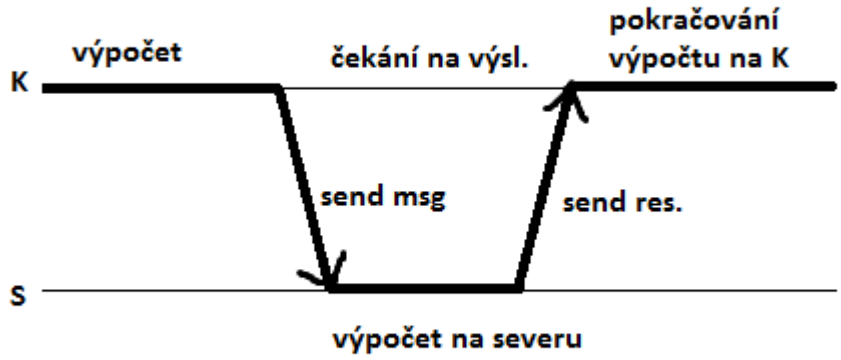
(d) Např. nějaký účetní program, který běží na klientovy a provádí změny a získává data z databáze

(e) Distribuovaná databáze

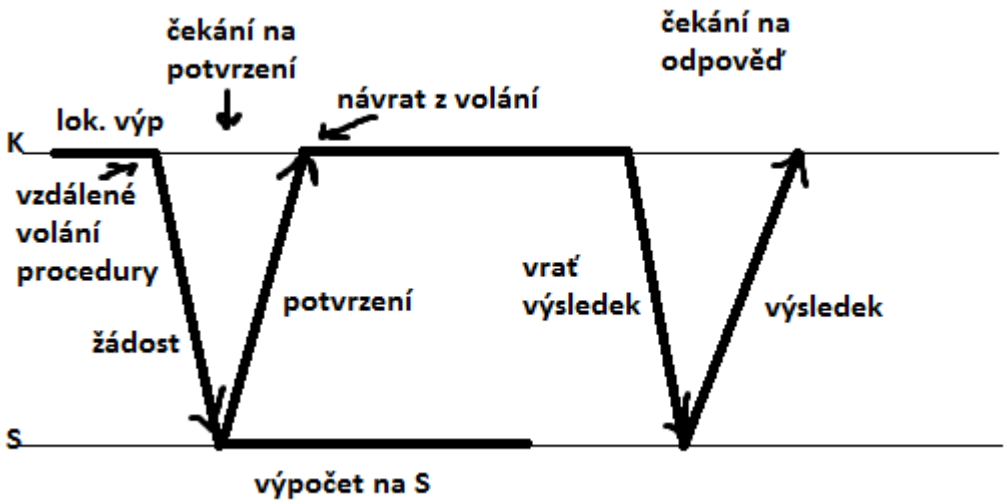
3. Nakreslete diagram komunikace mezi klientem a serverem s použitím synchronního a asynchronního posílání zpráv s potvrzováním.

(blokované / neblokované)

Synchronní



Asynchronní



4. **Co je to sémantika volání vzdálených podprogramů, jaké sémantiky znáte a čím se liší.**

   Cílem RPC systémů je vytvořit sémantiku volání pokud možno tak, aby se co nejméně lišila od sémantiky lokálního volání – některým odlišnostem se však vyhnout nelze

   Sémantiku zejména ovlivňuje možnost vzniku komunikačních chyb a chyb ve vzdáleném nebo místním uzlu. Různé systémy zpracovávají chyby různě, dostažní silné gramatiky může být složité. Nastane-li chyba, lze postupovat následovně:

   - Funkci voláme pouze jednou – nastane-li chyba, chybu nahlásíme nebo vyvoláme správce vyjímek, který zjedná nápravu
   - Neomezené čekání – po vyvolání fce budeme neomezeně dlouho čekat na její ukončení, což vede k zablokování procesu
   - Opakované volání – po uplynutí časového intervalu se volání opakuje

   Obecně rozeznáváme následující sémantiky (typy opakování volání):

   a) Provedení operace nejvýše jednou – funkce se vyvolá právě jednou, po vypršení časového kvanta se ohlásí chyba, operace nemusí být ve skutečnosti provedena ani jednou nebo právě jednou, nebo započata a nedokončena

   b) Provedení operace alespoň jednou – klient vyvolá fci a očekává její výsledek (potvrzení), nedostane-li do určité doby odpověď, opakuje volání funkce, je-li operace idempotentní (několikrát se provede akce a vždy stejný výsledek) odpovídá sémantika lokálnímu volání

   c) Provedení operace právě jednou – vyžaduje použití spolehlivé komunikace, v případě výpadku se jedná o závažnou chybu

   d) Získání výsledku posledním prováděním operace – blíží se sémantice lokálního volání, jako výsledek se chápe posledně prováděná operace

   e) Prováděním operace jednou nebo vůbec – tato sémantika předpokládá, že k provedení operace ve vzdáleném uzlu je využit transakční mechanizmus

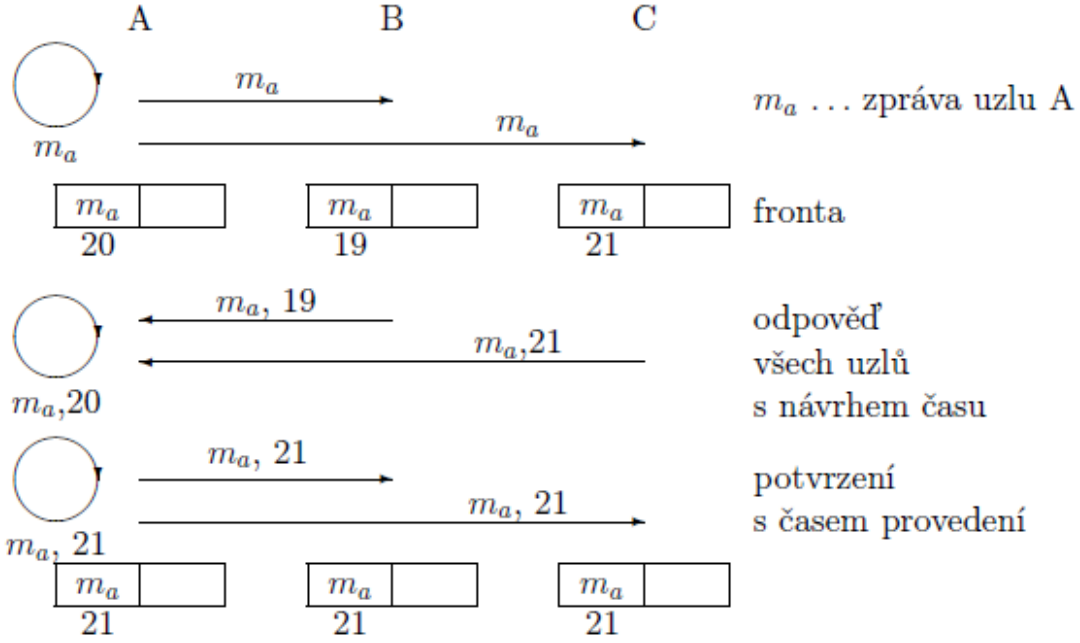
5. **Nakreslete schéma souběžného přenosu dvou zpráv mezi dvěma uzly pomocí protokolu ABCAST. Co protokol zaručuje.**

   - Zaručuje, že se požadavky vykonají ve stejném pořadí (ale nemusí to být ve stejném čase)

   Popis fungování

   - Iniciátor pošle zprávu s kódem operace všem členům skupiny

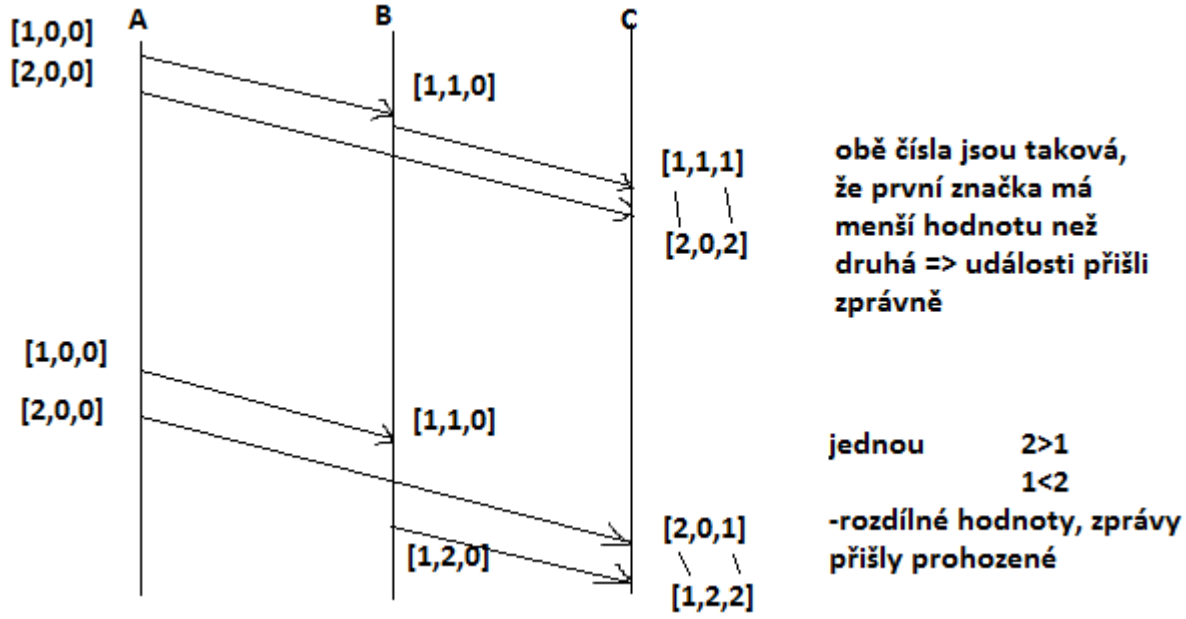
- Uzly odpoví zprávou s časovou značkou = návrh času provedení
- Iniciátor vybere **nejvyšší** hodnotu a odešle všem uzlům



Obrázek 4.7: Stanovení času provedení v protokolu ABCAST

6. Nakreslete schéma přenosu zpráv mezi dvěma uzly pomocí protokolu CBCAST.

(protokol s oslabeným uspořádáním) – existuje vztah příčina důsledek – zprávy generované jedním procesem mohou do cílového uzlu dorazit v různém pořadí, ale cílový uzel je musí zpracovat v původním pořadí => obecné řešení – zpráva přenáší svoji historii), např. čísla předchozích zpráv



7. Popište mechanizmus synchronizace hodin reálného času pomocí časových serverů.

8. Popište Christiansův algoritmus synchronizace hodin mezi dvěma uzly.

Získání běžného času z časového serveru. Christiansův algoritmus předpokládá, že časové zpoždění vzniklé při síťové komunikaci je stejné ve směru klient – server a naopak.Ve chvíli kdy klient zjistí, že hodlá požádat časový server o velitelský čas – zapamatuje si svůj aktuální čas. V okamžiku kdy přijde ze serveru odpověď (např. o x sekund později) přičte klient k přijatému času x/2 a takto získaný čas teprve označí jako svůj systémový.

Popřípadě udělám několik měření kde počítám (čas přijetí – čas odeslání – doba obsluhy)/2, takové delta T poté přičítám k přijatému času.

9. Popište Berkeley algoritmus synchronizace hodin reálného času.

- Je vybrán hlavní „master“
- Master se zeptá ostatních „slaves“ na jejich čas – podobnou cestou jako Christianův algoritmus
- Slaves odpoví
- Master zprůměruje časy, které nejsou mimo hlavní rozsah
- Master odešle slaves aktuální čas – kterým se mají nastavit

Předpokladem je aktivní časový server, který posílá periodické dotazy na čas klientů. Používá se v případě, že neexistuje přesný zdroj času. Slouží pouze ke srovnání času na spolupracujících serverch.

10. V zadaném obrázku doplňte hodnoty logických maticových hodin.

Logické hodiny = důležité je pořadí nikoliv čas = událost je časová značka, uspořádání událostí

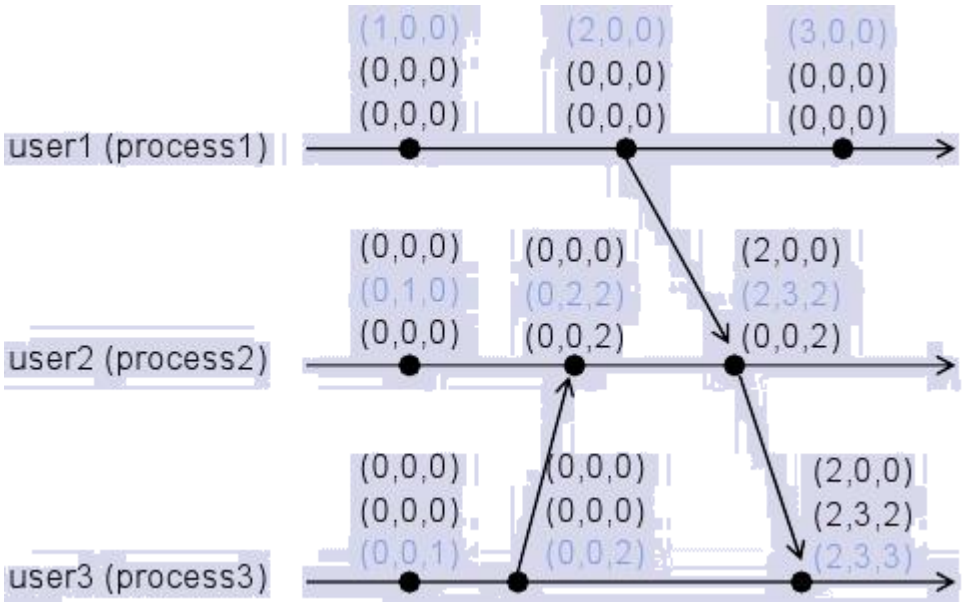
Vektorové hodiny = můj celkový pohled na události, zajišťují kauzalitu událostí, vektor délky n (n = počet uzlů), v každé složce čas daného uzlu

(a) Na začátku neznám stav ostatních => 0 a sobě dám 1

(b) Odesílání: svůj čas zvýším a odešlu

(c) Přijetí: svůj čas zvýším a vyberu MAX z porovnávání ze zbývajících složek vektoru

Maticové hodiny = pohled ostatních na události – každý proces si udržuje vektor s odeslanými zprávami (sem si ukládá při přijetí nějaké další zprávy od příjemce čas odeslání poslední zprávy od něj, kterou příjemce dostal) a zároveň co mu došli od všech ostatních. Z toho zjistí které zprávy už mají všichni (a jsou stabilní).



11. Co jsou to vektorové časové značky a čím se liší od časových značek dle Lamporta. V zadaném schématu označte vektorové časové značky a určete, které události jsou konkurentní a které ne.

Uspořádání událostí – Lamportův algoritmus

Logické hodiny = prostředek pro uspořádání událostí jiným způsobbem než podle absolutního času

Lamportovy časové značky – slouží k synchronizaci logických hodin - proces má čítač, který čísluje události, hodnota se určí tak že se vezme max ze salané hodnoty a hodnoty čítače a tato hodnota se zvýší o 1

Když proces A zašle zprávu procesu B a zhruba v témže okamžiku proces C zašle zprávu procesu D, pak jsou tyto zprávy konkurentní, neboť se v žádném případě nemohou ovlivnit. Jestliže se události *x,y* staly ve dvou procesech, které spolu nekomunikují, pak neplatí ani *x->y* ani *y->x*; říkáme, že události jsou konkurentní, tj. nedá se nic říci o tom, která událost se stala první

Vektorové číselné značky- události se ukládají do vektoru, pokud platí že všechny události Ui<Uj, pak události Ui předcházeli Uj, pokud jsou ve vektoru pro x1>y1 a x2<y2 jsou události konkurentní

Kauzalita není když: a->b pak čas(a)->čas(b), ale neplatí jestliže čas(a)< čas(b) pak a->b) **?!**

12. Co je to globální stav, co je to konzistentní řez.

- známe stav všech členů skupiny a přitom součet všech hodnot není < nebo > jak skutečné celkové prostředky

-algoritmus detekce globálního stavu pomocí chandy-lamportova algoritmu:

(a) iniciátor vyšle všem výstupním uzlům značku

(b) příjem první značky

- uzel si zapamatuje stav uzlu = poslední přijaté a odeslané zprávy = svůj stav
- stav všech příchozích kanálů označí jako prázdný
- vyšle všem výstupním uzlům značku

(c) příjem zprávy od uzlu, od kterého ještě značka nepřišla

- zapamatuje si čísla zpráv

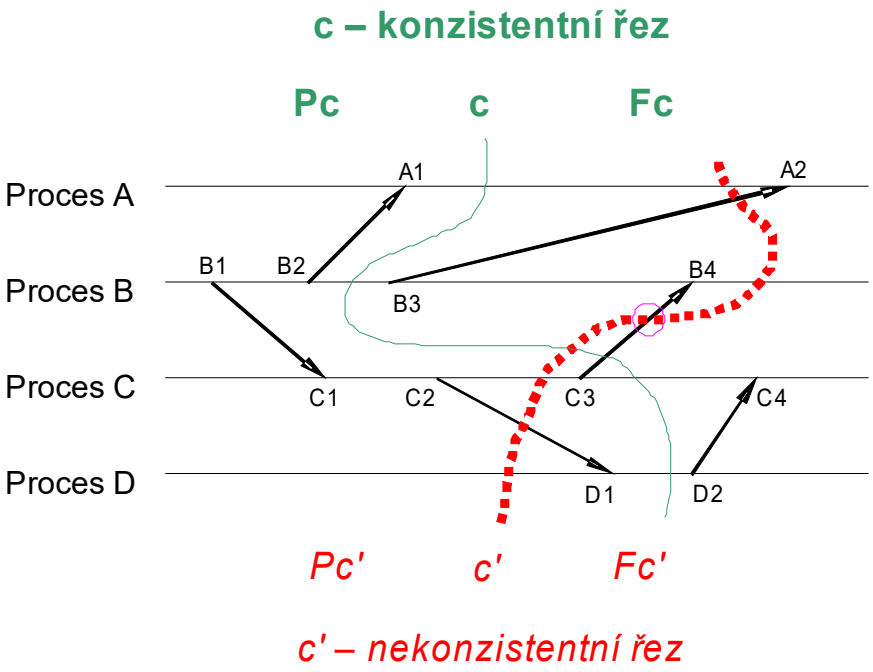
(d) příjem značek od dalšího uzlu

- zaznamenané zprávy došlé od uzlu mezi prvním a touto značkou = stav kanálu

(e) konec algoritmu po přijetí všech značek

Po přijetí všech značek konec, zapamatované stavy uzlů a kanálů definují konzistentní stav.

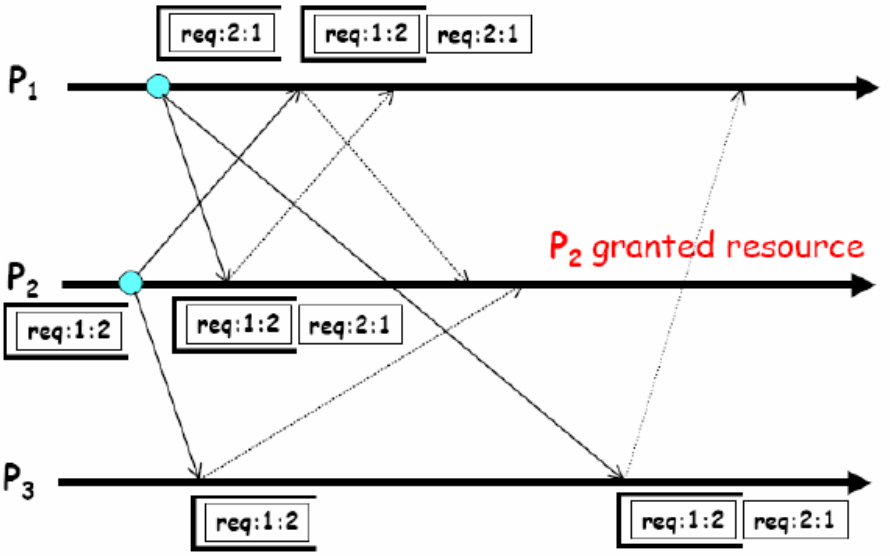
Konzistentní stav = jde po uzlech a jejich bodech kde proběh příjem první značky



13. Naznačte, jak bude fungovat algoritmus pro určení globálního stavu pro tři procesy s tím, že každý proces může komunikovat s oběma svými sousedy.

Postup dle příkladu 12

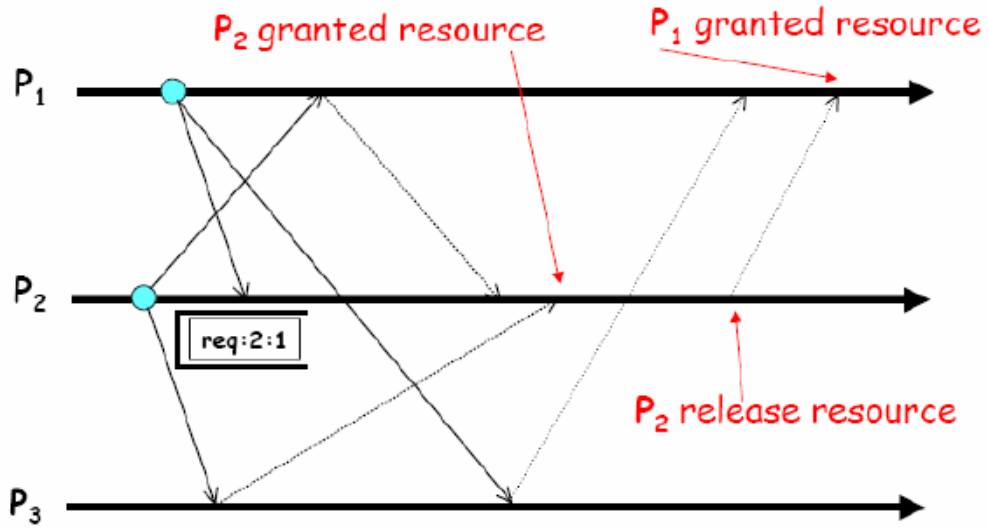
14. Naznačte, jak probíhá 3 fázový Lamportův algoritmus pro řešení úlohy distribuovaného vzájemného vyloučení s pomocí časových značek pro 2 procesy.



- Proces P požaduje zdroj posláním požadavku do všech procesů zasláním zprávy req.
- Při příjmu požadavku je požadavek zařazen do fronty požadavků uspořádané dle časových značek, poslání zptávy ach
- Ukončení zpracování požadavku – odstranění požadavku z fronty a poslání zprávy rel ostatním procesům
- Přijetí zprávy rel od procesu P – odstranění požadavku procesu P z fronty
- Povolení zpracování požadavku – požadavek je na prvním místě ve frontě a je potvrzen ostatními procesy
- [req:časová značka:proces]

15. Naznačte, jak probíhá 2 fázový Lamportův algoritmus pro řešení úlohy distribuovaného vzájemného vyloučení pomocí časových značek pro 2 procesy.

Ricard-Agrawale = optimalizace Lamportova algoritmu snížením počtu zpráv na 2(n-1)



- Proces P požaduje zdroj posláním požadavku do všech procesů zasláním zprávy req
- Při příjmu požadavku je požadavek potvrzen zprávou ach pouze tehdy, jestliže přijímací proces zdroj neužívá nebo o něj právě nežádá. Jinak je požadavek zařazen do fronty.
- Ukončení zpracování požadavku – poslání zprávy ach procesům, kterým bylo poslání předtím odepřeno (a odstranění požadavků z fronty)
- Povolení zpracování požadavku = požadavek je potvrzen ostatními procesy

16. Uveďte vlastnosti transakcí a vysvětlete je na příkladech.

a) Atomičnost – musí se dokončit celá transakce, pokud se to nepodaří dojde ke zrušení celé transakce – rollback – zajištěna konzistence

b) Konzistence – data jsou pořád v konzistentním (uceleném) stavu (po dokončení transakce - nemůže se tedy stát, aby například byla v systému faktura bez zákazníka

c) Izolace – data používaná během jedno transakce nemohou být využita současně v jiné transkaci, zrušení transakce nemá vliv na jinou

d) Trvanlivost – jakmile dojde k potvrzení transakce, její vliv přetrvává i po selhání systému

17. Co jsou to vnořené transakce a jak se zpracovávají.

- vnořené transakce jsou realizovány tak, že uvnitř transakce jsou vykonávány další transakce. Ty lze vykonávat paralelně – to dovoluje urychlit proces zpracování souběžným zpracováním vnořených transakcí. Další výhodou je možnost eliminovat velikost oblasti, ve které se projeví případná chyba

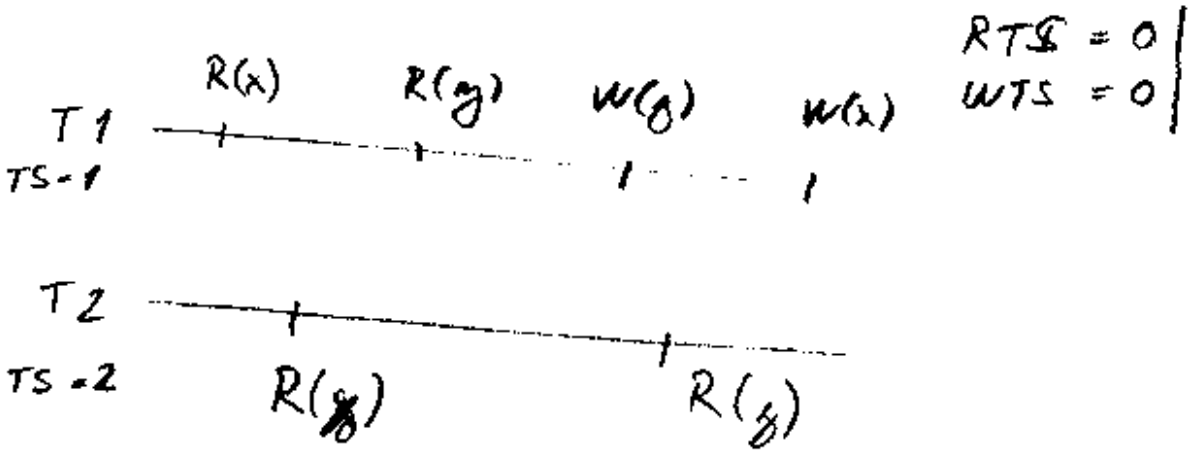
Vnořené transakce musí mít následující vlastnosti:

- Transakce nejvyšší úrovně může vyvolat více vnořených transakcí, které mohou pracovat paralelně. Synchronizace je zajištěna tak, že rodič se pozastaví dokud všichni jeho potomci neskončí nebo se nezruší
- Vnořené transakce mohou obsahovat zámky, které jsou ovládány rodičem, nikoliv sourozencem. To je rozumné protože rodič může být pozastaven pokud potomek pracuje.
- Pokud transakce skončí úspěšně, jsou všechny její zámky vráceny rodiči, který ji vytvořil (včetně zámků zděděných). K úplnému uvolnění zámků dojde až tehdy, když skončí

transakce na nejvyšší úrovni. To odpovídá situaci, kdy změny provedené vnořenými transakcemi budou platné až tehdy, když je spáchána transakce nejvyšší úrovně.

- Objeví-li se chyba, zruší se vnořená transakce dříve než dosáhneme konce. Vše je pak vráceno do původního stavů (operace vycouvání – undo), všechny získané zámky jsou uvolněny nebo vráceny rodičům, a jejich rodiče jsou o tom uvědoměni. Rodič se může sám rozhodnou, bude-li v transakci pokračovat nebo svou transakci také zruší

18. Jsou dány dvě transakce T1 {R(x); R(y); W(y); W(x) } a T2 {R(x); R(y)}. Naznačte algoritmus řízení souběhu pomocí časových značek.



19. Je dána transakce {z ← x; x ← y; y ← z }. Rozepište, co se uloží do logu a jak se obnoví data v případě zrušení transakce.

z:=x; x:=y;y:=z;

1) z = z | x
2) z = z | y; x=x|y;
3) z = z | y; x=x|y; y=y|z;

pokud x = 0; y= 1; z= 2;

4) z = 2 | 0
5) z = 2 | y; x=0|1;
6) z = 2 | y; x=0|1; y=1|0;

20. Co jsou to distribuované transakce. Popište 2-fázový commit.

Distribuované transakce. Jsou-li soubory umístěny v jednom uzlu, je ukončení transakce jednodušší v tom smyslu, že je znám globální stav. Jsou-li procesy provádějící transakce rozmístěny ve více uzlech, pak je třeba použité algoritmy modifikovat. V těchto případech se používá metoda vícefázového ukončování transakce. Nejčastěji 2 nebo 3 fázové ukončení. …

Problém ukončení nebo rušení distribuované transakce lze zvládnout pomocí dvoufázového ukončování. Zabezpečit, aby bylo provedení distribuované transakce odolné proti chybám technického vybavení je složitější. Nejcitlivějším místem transakce, kde může nastat chyba je spáchání transakce. Aby se eliminovala možnost přechodu databáze do nekonzistentního stavu používá se též protokol 3 fázového uzamykání.

Metoda 2 fázového ukončování. Podle způsobu komunikace lze rozdělit do 3 skupin:

(koordinátor musí uchovávat záznamy do doby než zjistí, že všichni sluhové ukončili transakci)

(a) Centralizované 2 fázové ukončení transakce. Provádění transakce v distribuovaném prostředí předpokládá existenci koordinačního servera, který transakci inicializuje a řídí její provedení a ukončení. 2 fázové ukončování transakce spočívá v tom, že v prvé fázi tento koordinační server inicializuje transakci a vyzve ostatní uzly (označené jako sluhy) k jejímu provedení. V druhé fázi čeká na odpovědi. Jsou-li všechny kladné, vydá příkaz ke spáchání transakce v ostatních uzlech. Je-li jedna z odpovědí záporná, vydá příkaz ke zrušení transakcí. Po vydání příkazu ke spáchání transakcí již koordinační server nepředpokládá, že by některý z ostatních serverů transakci zrušil. Pouze čeká na spáchání transakce z důvodu synchronizace.

(b) Lineární 2 fázové ukončování transakce. V případě lineárního ukončování transakce je provádění jednotlivých transakcí v jednotlivých uzlech zřetězeno. Koordinační sever vydá příkaz do druhého uzlu, atd. Nemůže-li být transakce provedena, nepředá příkaz k provedení, ale příkaz k zrušení. Dojde-li zpráva do posledního zřetězeného uzlu, je provedena kontrola máli se transakce provést. Nemá-li se transakce provést, předá se příkaz ke zrušení. Zřetězením v opačném směru projdou příkazy spáchej transakci nebo zruš transakci až ke koordinačnímu server. Ten provede/zruší transakci jako poslední.

(c) Distribuované 2-fázové ukončení transakce. Distibuované 2-fázové ukončení transakce začíná opět akcí koordinátora, který transakci inicializuje. Rozhodnutí o tom, bude-li transakce provedena nebo zrušena však neposílají ostatní uzly koordinátoru, ale sobě navzájem.

21. Popište realizaci operací P a V nad semafory s využitím nedělitelných operací advance(E), await(E,v) a ticket(S).

Čítače událostí – jsou celočíselné proměnné, nabývajících nezáporných hodnot, nad kterými jsou definovány operace:

- Advance – nedělitelně zvyšuje hodnotu proměnné o jedničku
- Read – čte hodnotu proměnné
- await – čeká, pokud je uvedená hodnota menší než hodnota čítače událostí
- Ticket – vrátí starou hodnotu sequenceru a současně zvýší hodnotu sequeneceru o jedničku

V(sem) <-> advance(sem.E) //zvýšení hodnoty E

P(sem) <-> v = ticket(sem.S); //vydání lístku

await(E,v); //čekání dokud se číslo globální proměnné nerovná lokální proměnné v (~while/v<E)

22. Jaký rozdíl je mezi striktní a sekvenční konzistentností. Ukažte na kontra příkladech.

Striktiní konzistence – jakékoliv čtená z paměti z adresy *x* vrátí hodnotu uloženou při posledním zápisu na adresu *x*. Všechny zápisy jsou okamžitě všude viditelné, musí existovat přesný globální čas. Ideální pro programování, v distribuovaném systému však nedosažitelné.

Příklad:

a=1;a=2; print(a) //vytiskne vždy 2

Striktní konzistence: P1:W(x)1 poté P2:R(x1)

Paměť, která není striktně konzistentní: P1:W(x)1 poté P2:R(x)0 R(x)1

Sekvenční konzistence – jestliže procesy běží na různých procesorech, je povoleno libovolné prokládání jejich instrukcí, avšak s podmínkou, že všechny procesy vidí stejné pořadí změn paměti. Změny nejsou propagovány okamžitě, nic se nemluví o velikosti zpoždění, je pouze zaručena kauzalita.

Příklad:

	Sekvenčně konzistentní paměť					**Paměť, která není sekvenčně konzistentní**				
P1:	W(x)a					W(x)a				
P2:		W(x)b					W(x)b			
P3:			R(x)b		R(x)a			R(x)b		R(x)a
P4:				R(x)b	R(x)a				R(x)a	R(x)b

Rozdíl mezi sekvenční a lineární k.je že lineární k kromě sekv. Kauz. Pracuje i s časovými značkami

23. Na obrázku ilustrujte, jak se od sebe liší sekvenční konzistentnost od FIFO konzistentnosti.

FIFO konzistentnost (také PRAM nebo Pipeline RAM)

(a) Zápisy od jednoho procesoru jsou ostatními viděny v témže pořadí, ve kterém byly vyslány

(b) Zápisy od různých procesů mohou být viděny různými procesy v různém pořadí

Příklad:

	Platná sekvence událostí FIFO konzistentnosti						
P1:	W(x)a						
P2:		R(x)a	W(x)b	W(x)c			
P3:					R(x)b	R(x)a	R(x)c
P4:					R(x)a	R(x)b	R(x)c

P1:a=1;if(b==0)kill(P2);

P2:b=1;if(a==0)kill(P1);

V případě PRAM (FIFO) konzistence je možný výsledek, že oba procesy budou zabity, a to v případě, že P1 přečte *b* dříve, než uvidí jeho zápis *a* P2 přečte dříve, něž uvidí jeho zápis. Tento výsledek není možný při libovolném uspořádání instrukcí, neodpovídá tedy sekvenční konzistenci, avšak vzhledem k PRAM(FIFO) konzistenci je korektní.

24. Co je to PRAM (Pipelined RAM) konzistentnost. Určete možné (správné) výsledky v zadaném příkladě.

Př. 23

25. Popište jak se provádí operace write-update a jak write-invalidate. Na obrázku naznačte, jak lze zajistit pomocí write-invalidate sdílení dat v případě, že vyžadujete vícenásobný přístup pro čtení a výhradní přístup pro čtení a zápis.

Existují 2 základní možnosti ja propagovat aktualizaci dat z jednoho procesoru ostatním.

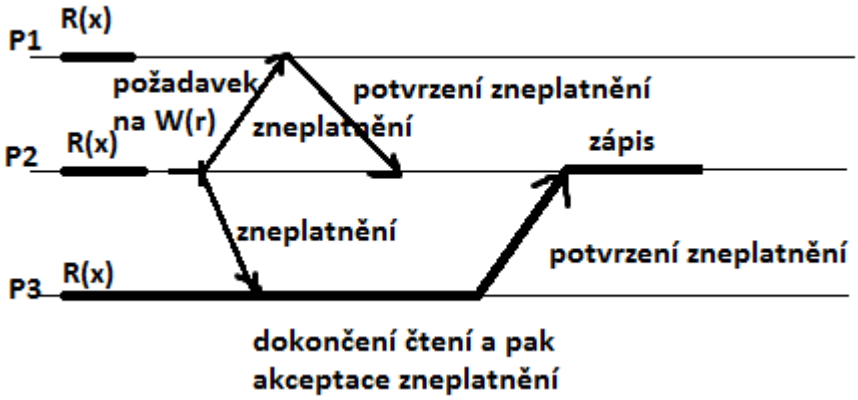
Write-update = aktualizace dat se provádí lokálně a pomocí skupinového adresování se rozesílá všem managerům replik, udržující kopie datových položek. Ti okamžitě modifikují data čtená lokálními procesy. Procesy čtou kopie lokálních dat bez potřeby komunikace. Kromě toho, že může existovat více čtenářů, může také několik procesů zapisovat tatáž data ve stejnou dobu.

Model konzistentnosti paměti, který je realizován s metodou write-update závislá na mnoha faktorech, zejména na vlastnostech skupinového protokolu. Sekvenční konzistentnosti může být dosaženo použitím skupinového protokolu s úplným uspořádáním, kdy se všechny procesy dohodnou na pořadí aktualizací. Výhodou jsou laciné operace čtení, které probíhá lokálně. Nevýhodou je relativně vysoká cena implmentace uspořádaného multicastu programově. Proto

v konkrétních realizacích se používají skupinové protokoly s technickou podporou na přístupové úrovni.

Write-invalidate = *obecně realizovaná v systémech typu multiple-read/single-writer sharing. Data mohou být buď zpřístupněna více procesům najednou pro čtení, nebo jednomu procesu pro čtení i zápis. Položka, která je zpřístupněná pouze pro čtení může být libovolně kopírována do ostatních procesů*. Pokud se proces pokusí zapsat do položky, pošle se nejprve do všech procesů zpráva, která ji zneplatní a teprve po potvrzení zprávy může proces položku aktualizovat. *Tím jsou procesy ochráněny před čtením starých dat. Všechny procesy, které se ve stejnou dobu pokusí aktualizovat položku jsou blokovány do doby, než je aktualizace vybraným procesem dokončena. Výsledkem je, že procesy přistupují k položce při zápisu uspořádaně – podle pravidla první přijde, první je obsloužen.Toto přístupové schéma zajišťuje sekvenční konzistentnost.*

Pokud se použije zneplatnění zpožděné, dosáhneme repase-consistency.



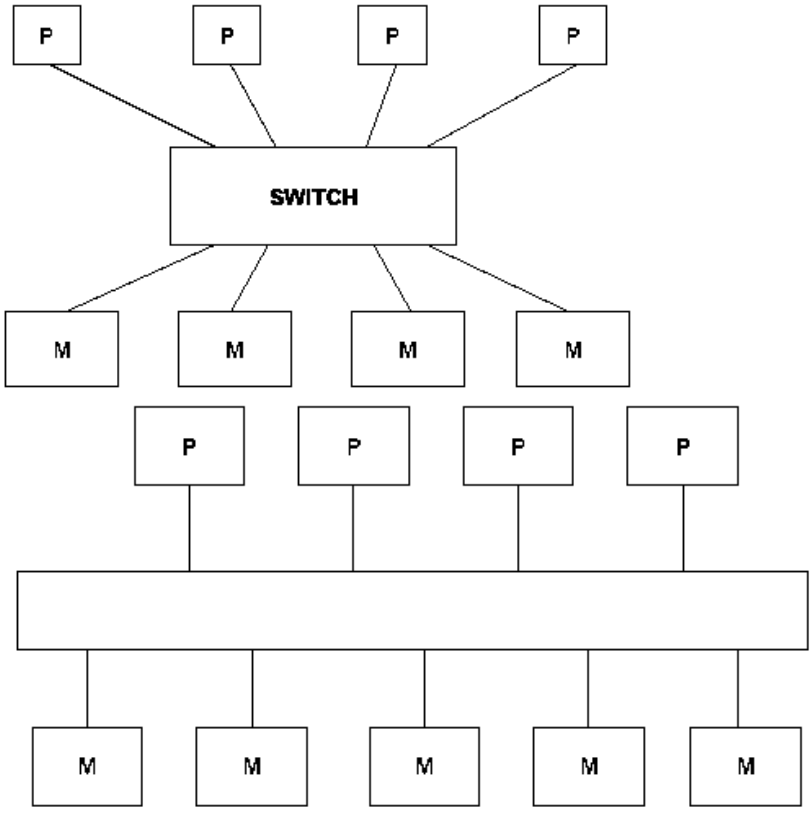
26. Co je to distribuovaná sdílená paměť, jak je organizována a jaké jsou technické předpoklady pro její realizaci. Uveďte některé problémy.

Distribuovaná sdílená paměť (DSM) je abstrakce používaná pro sdílení dat mezi procesy v počítačích, kde není sdílená fyzická paměť. Uživatel nemusí explicitně řešit problém přenosu dat mezi uzly. Základním problémem je dobrá propustnost komunikačního subsystému, která by neměla záviset na rozlehlosti systému a počtu procesorů. DSM je prostředek pro řešení paralelních aplikací, distribuovaných aplikací nebo skupinových aplikací, ve kterých jsou individuální sdílené datové položky přístupné přímo. Jedná se o model typu peer-to-peer, nikoliv model klient/server. Data jsou vyměňována komunikačním systémem. V každém uzlu je z důvodu rychlého přístupu lokální kopie dat, systém musí zajistit konzistentnost dat v jednotlivých uzlech

- Konzistentní modely (striktní konzistence, sekvenční,…)
    - Konzistenční modely bez použití synchronizačních proměnných
        - implementace možná na úrovní virtuální paměti
        - procesy nemusí o DSM vůbec vědět
        - standardní programovací jazyky / knihovny
    - Konzistenční modely s použitím synchronizačních proměnných
        - speciální jazyky nebo knihovny
        - procesy musí být korektně naprogramovány
        - vyšší výkonnost
- Distribuované stránkování (obdoba virtuální paměti) – základní schéma je obdobou virtuální paměti – je-li referencována stránka, která není na lokálním stroji, je vyvoláno přerušení a stránka musí být načtena ze stroje, kde se právě nalézá. Problémy: replikace => konzistence stránky, správa kopii, falešné sdílení.

Jestliže jsou stránky (především z důvod výkonnosti) replikovány, je třeba udržovat konzistenci dat. Typická implementace je namapování všech stránek read-only a v případě zápisu provádět potřebné synchronizační akce. Teoreticky jsou možné dva způsoby – invalidce nebo aktualizace. Distribuovaná stránkování v naprosté většině používá invalidaci – vzhledem k velikosti přenášených dat a době přenosu.

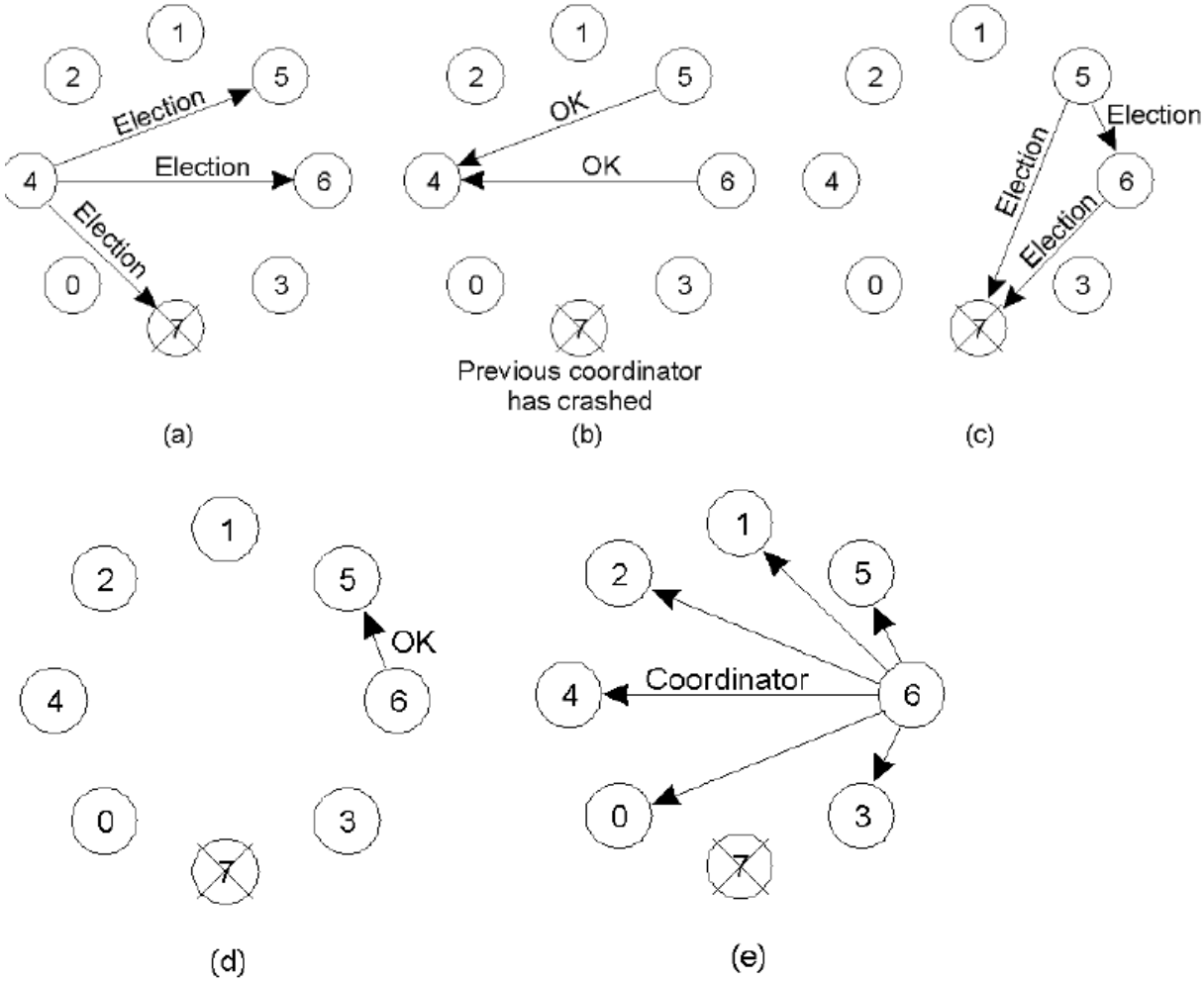
- Distribuované sdílené proměnné –implementace na úrovni knihoven. Výhody: potenciálně lepší výkonnost, eliminace falešného sdílení. Nevýhody nepodporování přímo operačním systémem, nutnost implementace pro různé jazyky.
- --------
- => Soubor počítačů sdílí 1 adresářní virtuální prostor
- Založeno na principu stránek nebo sdílených proměnných
- Odkazy na lokální stránku jsou realizovány hardwarem, odkazy na vzdálené stránky způsobí výpadek stránky a str. se natáhne ze vzdáleného adresáře
- Problémy s replikacemi stránek (pouze 1 kopie – neefektivní, jednoduché,read only, /rw – čtení – nastavení kopie na read only, zápis = zneplatnění nebo oprava)
- Problém jak najít všechny kopie pro zneplatnění: buď vysílat všem (broadcast – přeruší i ty co stránku nevlastní, neefektivní využití šířky pásma) nebo manager stránek
- Sdílené proměnné – synchronizace pomocí zámku, podmínek a bariér – uloženy do zvláštní stránky
-



27. V zadaném příkladě popište výměnu zpráv algoritmus vhazování (Bully algoritmus).

- Koordinátorem se má stát proces s nejvyšším ID =>
- Nějaký proces zašle zprávu procesům vyšším ID (a)
- Pokud přijde nějaká odpověď – vzdává se (b) (d)
- Pokud nepřijde nic, stává se novým koordinátorem a zašle zprávu o výsledku všem (e)

- Při přijmu zprávy o volbě každý proces zašle žádosti všem vyšším procesům (c) (volba ve dvou kolech)
- Při překročení timeoutu možnost více koordinátorů



28. Popište decentralizovaný algoritmus vzájemného vyloučení s použitím pověření, které je předáváno v logickém kruhu.

Vzájemné vyloučení = (a) proces 1 žádá koordinátora o povolení vstoupit do kritické sekce a dostává povolení. (b) poté žádá o povolení vstoupit do téže kritické sekce proces 2 – koordinátor neodpovídá. (c) když proces 1 opouští kritickou sekci, oznámí to koordinátorovi a ten odpoví procesu 2

Distribuované vzájemné vyloučení - Decentralizovaná metoda, založena na předávání pověření.

Logický kruh:

- Procesy jsou uspořádány v kruhu podle PID
- Proces, který zjistí, že koordinátor nereaguje, pošle zprávu se svým PID následovníkovi
- Ten do ní přidá svoje PID
- Jakmile dojde zpráva k procesu, který volbu začal, tak tento proces vybere nejvyšší PID
- Po vybrání nejvyššího PID se nechá kruhem kolovat zpráva COORDINATOR s vybraným PID (tohle je centralizovane, necentralizovaně →podčarou)

- Procesy jsou uspřádány do logického kruhu
- Na začátku má pověření proces č. 0

    - Pověření se posílá v kruhu – dvoubodové spojení
    - Po obdržení pověření může proces do KS po opuštění KS předává pověření dále
- Problém s detekcí ztráty pověření
- Jednoduché rozšíření kruhu

29. Popište decentralizovaný algoritmus vzájemného vyloučení s použitím pověření, které je předáváno ve stromové struktuře.

Distribuované vzájemné vyloučení - Decentralizovaná metoda, založena na předávání pověření.

Strom

- Iniciátor volby rozešle WAKE UP zprávu, čímž dá jasně najevo, že začíná volba
- Volit se začíná od listů, každý list vyšle svoje PID k rodiči
- Jakmile rodiči dojdou všechny PID potomků vybere min (z PID rodiče, PID potomků) a ten pošle rodiči
- Nakonec ke kořeni vybublá nejnižší PID ve stromu a ten se stává koordinátorem
- (tohle je centralizovane, necentralizovaně →podčarou)

- Procesy uspořádány do stromové struktury
- Proces udržuje frontu požadavků na pověření od procesů nižší úrovně
    - Požaduje-li vstup do KD a fronta je prázdná, posílá požadavek svému rodiči a řadí se do fronty
    - Požaduje-li podřízený proces vstup do KS, předá mu pověření nebo řadí do fronty
    - Obdrží-li pověření, předá ho prvnímu ve frontě

30. Popište algoritmus pro detekci ukončení úlohy.

- Proces uchovává představu o potomcích ve stromové struktuře
- Ví kolik procesů bylo vytvořeno
- Když nějaký proces dokončil svoji činnost, signalizuje to rodiči => ví se kdy skončil poslední proces

Synchronní systémy

- Dijkstra-Scholten – difuzní výpočty, uspořádání do stromu, iniciátor je kořen, detekce ukončení podle počtu ukončených podřízených
- **Dijkstra-Scholten algoritmus**
    - Jestliže graf procesů je strom, pak každý listový proces při přechodu do stavu ukončen pošle signál svému otci
    - Jestliže proces dostane signál od všech svých synů, pošle signál svému otci
    - Jestliže všechny signály dostane inicializační proces, je distribuovaný výpočet ukončen
- Shavi-Francez – více iniciátorů, všechny procesy se účastní vlny, proces který není iniciátorem pokračuje ve vlně,pokud strom kolabuje – iniciátory pokračují ve vlně

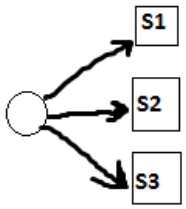
31. Co jsou to vícenásobné kopie. Uveďte základní algoritmy údržby vícenásobných kopií.

Vícenásobné kopie – data, která se v distribuovaných systémech vyskytují ve více kopiích, urychlují tak činnost (čtení) – slouží k rozložení zátěže

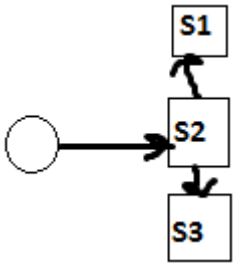
- Zvyšují výkonnost a redukují dobu přístupu
- Zvyšují také režii pro udržení konzistentnosti
- Zvýšení dostupnosti rozmisťováním kopií do různých uzlů sítě
- Pokud počet čtení << zápisy => vysoká konzistentnost zvyšuje režii

Údržba – zajištění konzistence ve vybraném stupni konzistentnosti

- Aktivní replikace (explicitní) uzel sám posílá data do ostatních uzlů (uživatel se sám stará o udržování konzistence)



- Pasivní replikace – uzel se ptá, jestli nedošlo ke změně – pokud ano aktualizace (odložená r. – zápis do primární r., aktualizace replik později)



32. Jaký je rozdíl mezi klient-centric a data-centric systémy. Co je to eventual consistency, kde se používá.

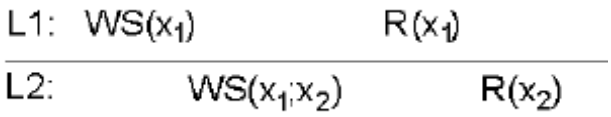
Data-centric konzistentní modely (uvolněná, slabá, fifo, příčinná, sekvenční, linearizovatelnost, striktní konzistence). Nepožaduje se stejný pohled na data pro všechny klienty, ale aby se jevily data konzistentně jednomu klientovi s ohledem na jeho operace. (monotónní čtení & zápis, čtení vlastních zápisů, zápisy následující po čtení)

Klient – Centric konzistenční modely. Předchozí (uvolněná, slabá, fifo, příčinná, sekvenční, linearizovatelnost, striktní konzistence,slabá,uvolněná,vstupní) studované modely se týkaly údržby konzistentní datové paměti v případě konkurentních operací zápisu a čtení. Jiná třída distribuovaných datových pamětí je charakterizována tím, že postrádá souběžné opravy. Tj. mnoho operací požaduje čtení datové paměti, ale ne zápisy. Takové datové paměti používají velmi slabou formu konzistentnosti, známou jako možná, konečná konzistentnost.

Možná (Eventual) konzistentnost. Mezi systémy, kde může být velmi uvolněná konzistentnost patří: DNS systém, web. Otázka: jak rychle mohou být opravy přístupné procesům, které data pouze čtou? – po nějaké době vidí všechny repliky všechny opravy a přecházejí do konzistentního stavu (jsou postupně konzistentní). Tento typ konzistentnosti je znám jako *možná (konečná) konzistentnost*. Možná (konečná) konzistentnost vyžaduje zadávat pouze takové opravy, u kterých je zaručeno, že budou v konečné době doručeny do všech replik. Možná (konečná) konzistentnost pracuje dobře, pokud se klienti vždy připojují k téže replice. Pokud se do distribuovaných systémů přidá mobilita, může systém zkolabovat velmi rychle.

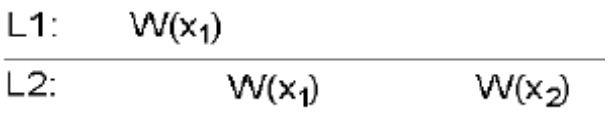
33. Co to je monolitic-read a monolitic-write. Jak to souvisí s eventual consistency.

(asi myšleno monotónní čtení a zápis ?!) mr&mw = data-centric konzistentní modely = nepožaduje se stejný pohled na data pro všechny klienty, ale aby se jevily data konzistentně jednomu klientovi s ohledem na jeho operace



Monotónní čtení – operace čtení jsou prováděny jedním procesem *P* na dvou různých lokálních kopiích téže datové paměti.

- po přečtení hodnoty x všechna další čtení vrátí stejnou nebo novější hodnotu (při připojení k jiné replice uživatel vidí všechny zprávy co už si přečetl dřív – dá se řešit třeba logem updatů, co už klient viděl – replika si pak si může ověřit, že je dost aktuání případně si sehat aktalizaci a poskytnout správná data.



Monotónní zápis – operace zápisu prováděné jedním procesem *P* nad dvěma různými lokálními kopiemi téže datové paměti

- zápis do proměnné je proveden před každým následním zápisem do ní; než do repliky zapíšu, musí si aplikace přijmout aktuální změny od ostatních. Implementace: podle klientského write-setu si replika ověří, jestli něco nemá dozapsat, po zápisu si klient aktializuje write set.

Eventual consistency –ot. 33 – souvislost ?

34. Co je to read-your-writes a writes-follow-reads a jak to souvisí s eventual consistency.

klientocentrické konzistenční modely
read-your-writes = procesy při následném čtení vidí svoje zápisy (po aktualizaci wiki nekoukám na kopie z cache). Implementace: buď forward čtení na aktuální repliku, nebo replika ověřuje podle write-setu svoji aktuálnost.
Zápis proměnné je proveden před jakýmkoli následným čtením této proměnné
Writes-follow-reads = zápis se provede do kopie proměnné, která je alespoň tak aktuální jako ta, která se předtím přečetla. Implementace: aktualizace podle read setu. Po zápisu se aktualizuje read-set i write set klienta.
Zápis proměnné po předchozím čtení této proměnné je proveden na stejné nebo novější hodnotě
Eventuální konzistence = po ukončení všech zápisů budou všechny repliky v konečném čase aktualizovány; problém je, že jeden proces může koukat na data jiných replik a vidět něco jiného

35. Naznačte algoritmus detekce uvíznutí v distribuovaném systému se třemi oblastmi.

36. Co je to uvíznutí, jaké jsou nutné podmínky pro uvíznutí, jak se řeší detekce a odstranění uvíznutí v distribuovaných systémech.

Soubor procesů, blokovaných čekáním na podmínky, které nemohou nastat. Ilustrace grafem WFG (wait for graph)

Podmínky vzniku uvíznutí

- Vzájemné vyloučení – zdroj je buď dostupný, nebo výhradně přiřazen právě jednomu procesu
- Hold and wait – proces držící váhradě zdroje může požadovat další
- Nemožnost odejmutí
- Cyklické čekání – každý čeká na zdroj držený jiným členem

Detekce distribuovaného deadlock. Sledování WFG nebo RAG

Centralizovaný algoritmus – koordinátor udržuje globální WFG a hledá cykly – je jednoduchý

Distribuovaný algoritmus – globální WFG se schopností detekce rozšířené na více uzlů

Hierarchický algoritmus – hierarchická organizace, strany detekují deadlock pouze zahrnutím svých potomků

Odstranění 1) přerušení zacyklení grafu 2) (detekce a zotavení) přerušení jednoho nebo více procesů a vrácení jejich zdrojů 3) procesy se mohou vracet k synchronizačnímu bodu 4) ignorování 5)systematické zabránění pomocí pečlivé alokace zdrojů – bank. Alg. 6) prevence bránění vzniku jedné z podmínek pro vznik deadlocku - (výlučný přístup – vizualizace; přidělení všech zdrojů nebo požadavek na další; neumožněné odejmutí.)

RAG = ressource allocation graph

37. Co je to Bankéřův algoritmus, co řeší a jak.

Algoritmus plánování, který se dokáže vyhnout uvíznutí

Příklad: bankéř má 4 zákazníky A,B,C,D každému garantuje pujčku (6,5,4,7) = 22 dohromady; bankéř ví, že nebudou chtít všichni pujíčku současně, proto si pro obsluhu nechá jen 10. A předpokládá že při dosažení maximální pujíčky zákazník vrátí celou částku.

Zákazník	Má	Max
A	1	6
B	1	5
C	2	4
D	4	7

=>volné prostředky 10-(1+1+2+4)=2 => stav je bezpečný, protože lze pozastavit všechny požadavky kromě C (C dostane 2 jednotky => dosáhne tím maxima = 4 => skončí…uvolní tak 4 jednotky a ty lze půjčit B nebo D atd.),pokud ale bude:

Zákazník	Má	Max
A	1	6
B	2	5
C	2	4
D	4	7

=>10-(1+2+2+4)=1 => stav není bezpečný, pokud budou chtít všichni max. pujíčku bankéř neuspokojí nikoho a dojde k uvíznutí

!uvíznutí nemusí nastat sice nutně, s tím ale nelze počítat!

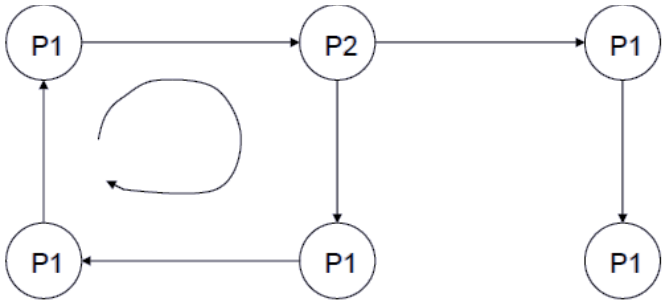
Rozhodování bankéře:

- U každého požadavku – zda vede k bezpečnému stavu
- Bankéř předpokládá, že všechny požadovaný zdroj byl procesu přiřazen a že všechny procesy požádaly o všechny bankéřem garantované zdroj
- Bankéř zjistí, zda je dostatek zdrojů pro uspokojení některého zákazníka pokud ano – předpokládá, že zákazníkovy byla suma vyplacena, skončil a uvolnil (vrátil) všechyn zdroje
- Bankéř opakuje krok 2, pokud mohou všichni zákazníci skončit ve stavu bezpečný

38. Co je to distribuovaný deadlock. Uveďte co je to OR model, AND model a P z Q model distribuovaného dreadlocku.

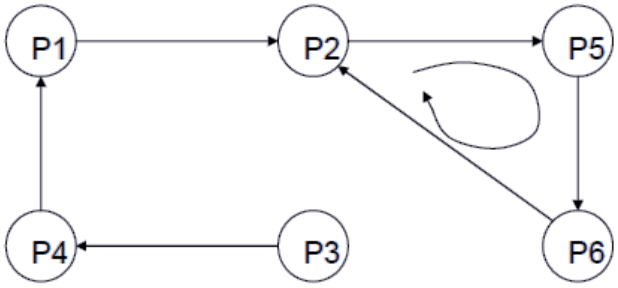
Deadlock může být distribuovaný, tedy obsahovat proces čekající na událost nebo prostředek na jiném počítači. Detekce distribuovaného deadlocku je ještě složitější než detekce dreadlocku na jednom počítači. => soubor procesů, blokovaných čekáním na podmínky, které nemohou nastat

AND model = proce může současně požadovat více zdrojů. Zůstává zablokovaný dokud neobdrží všechny požadované zdroje. Cyklus WFG (wait-for-graf) je postačující podmínkou



OR model = proces může najednou požadovat více zdrojů. Zůstává blokován dokud neobdrží alespoň jeden z požadovaných zdrojů. Nalezení uzlu v grafu (knot) je postačující podmínkou pro detekci deadlocku.

Knot (uzel) = podmnožina orientovaného grafu taková, že počínajíc z libovolného uzlu podmnožiny je nemožné opustit knot po hranách grafu.



P-out-of-Q model =získání P z Q zdrojů, například k replikám (stačí přístup k P replikám z Q replik).Speciální případy P=1 ..OR model, P=Q …AND model

39. Co je to distribuovaný systém souborů, uveďte příklady.

Síťové souborové systémy (*network filesystem*) je označení pro systémy souborů, které jsou dostupné prostřednictvím počítačové sítě. Ve skutečnosti leží soubory a adresáře na jiném počítači a přistupujeme k nim pomocí speciálních síťových volání služeb (např. SMB, NFS, CODA apod.). Na vzdáleném počítači jsou pak soubory a adresáře fyzicky uloženy v podobě klasického systému souborů. Speciálními síťovými systémy souborů jsou **distribuované souborové systémy** (např. GFS v Linuxu), které se mohou rozkládat na několika počítačích, které jsou navzájem propojeny pomocí počítačové sítě.

Při realizaci systému souborů je od sebe třeba odlišit souborové služby a souborový server. Souborové služby jsou dostupné funkce. Akce, které jejich volání způsobí, neříkají nic o jejich realizaci. Souborový server je proces, který realizuje souborové služby.

Distribuovaný systém souborů navíc od sebe odděluje souborové služby a adresářové služby. Souborové služby realizují manipulace se soubory, jakou jsou čtení, zápis, připojení,… Adresářové služby manipulují s adresáři. Vytváří a ruší adresáře, vytváří, ruší, mění jména souborů.

Příklady distribuovaných systémů

NFS(Network File Systém) lze chápat jako distribuovaný souborový systém. SUN Microsystems. Poskytuje transparentní vzdálený přístup ke sdíleným souborovým systémům prostřednictvím počítačové sítě. Je to realizováno kombinací funkcí jádra na straně klienta a NFS serveru. NFS protokol byl navržen tak, aby byl nezávislý na typu počítače, operačním systémů, síťové architektuře a transportním protokolu. Tato nezávislost je zajištěna pomocí mechanizmu volání vzdálených procedur. Dále obsahuje doplňky nebo moduly pro přidělování přístupových práv, možnost připojit vzdálený strom adresářů do určitého místa v lokálním souborovém systému, jednotnou administraci jmen a hesel uživatelů, uzamykání souborů…

AFS = (Andrew File Systém) distribuovaný souborový systém, dostupný jako základ pro DFS(distributed file systém). Cílem při vytváření bylo: maximální unifikace pracovního prostředí, nezávislost uživatele a souborů; transparentnost pro uživatele v UNIXu; spolehlivost, odolnost proti poruchám; velký výkon – nasazení v univerzitním prostředí velkého rozsahu s možností libovolného růstu.

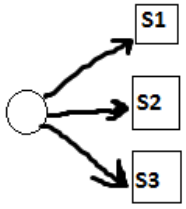
AFS versus NFS

AFS má jmenný prostor souborů, NFS je závislé na připojení. Soubory AFS mohou být libovolně rozmístěny po serverech aniž by se na stanici musely měnit body připojení. Použitelnost NFS silně klesá s rozsahem instalace. AFS vzhledem k použitému mechanizmu vyrovnávací paměti zvládne velké rozšiřování. AFS má zabudovaný mechanizmus bezpečnosti spolu s dalšími prostředky řízení přístupových práv. AFS umožňuje replikaci dat označených jako data pro čtení. Tím se zvyšuje spolehlivost systému a zlepšuje dostupnost dat. AFS dává možnost zálohovat data bez přerušení jejich přístupnosti. Systém informační databáze, oddělený od vlastních dat dává možnost přesunování AFS svazků mezi AFS servery bez přerušení jejich přístupnosti a nutnosti dalších zásahů. Oproti NFS zlepšuje možnost rekonfigurace. AFS lze rozdělit z hlediska správy do zcela autonomních oblastí a správu provádět z libovolné stanice. Pro přístup na AFS pomocí NFS existuje translátor AFS/NFS.

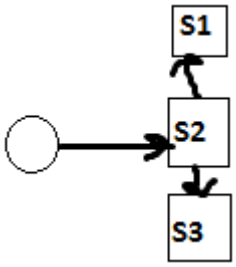
Další např. DCE, Amoeba (distribuovaný operační systém)

40. Co je to aktivní a pasivní replikace souborů.

Aktivní replikace (explicitní) uzel sám posílá data do ostatních uzlů (uživatel se sám stará o udržování konzistence)



Pasivní replikace – uzel se ptá, jestli nedošlo ke změně – pokud ano aktualizace (odložená r. – zápis do primární r., aktualizace replik později)



41. Vyjmenujte a popište sémantiky sdílení souborů v distribuovaném souborovém systému.

Sekvenční sémantika – sémantika používaná v UNIXu, výsledekm je poslední zapsaná hodnota

Relační sémantika – změny v souboru jsou pro ostatní viditelné po jeho uzavření (v otevřeném souboru jsou změny viditelné pouze procesu, který jej modifikuje)

Immutables files (neměnitelné soubory) – sémantika používaná v AFS – výsledkem je nová kopie souboru

Transakční sémantika – provedení operací čtení a zápisu jako by to byla transakce – nejnáročnější velká režie

42. Jak se provádí mapování souborového systému a jaká pravidla by se měla dodržovat. Jak se zde projeví transparentnost umístění a transparentnost přemisťování.

- Mapování zvdálených souborových systémů může být realizováno na každém z klientů jinak – mapování závislé na klientovi

Transparentnost umístění: v názvu cesty by nemělo být např. jméno serveru na kterým je soubor(y) uložen(y) – jinak problém při přesunu jinam

V přístupové cestě by nemělo být uvedeno jméno souboru – mapování na zvláštní svazek nebo do podadresáře – jméno souborového serveru musí být uživatelsky skryto, konfig. soubor – není odolné vůči přemístění, adresářový server – modifikace jména serveru mimo klienta – transparentní umístění

43. Vysvětlete problém osiřených stromů v distribuovaných souborových systémech.

- Link = Prostředek pro vytváření spojení mezi adresáři – dovoluje vytvářet obecné adresářové struktury, to ale vede k možným problémům

- Rozpad podgrafu na dva nesouvislé grafy,vznik odstraněním podadresáře, která je spojen přes další adresáře sám se sebou – u centralizovaných systémů musí ke každému adresáři existovat cesta ke kořenovému – v distribuovaných systémech ale problém

44. Popište, jak obecně funguje algoritmus shody. Ilustrujte na příkladu 3 procesů, které se mají dohodnout na jedné (minimální) hodnotě.

Algoritmus shody
- Všechny procesy Pi začínají ve stavu „nerozhodnutý“
- Každý Pi navrhne hodnotu Vi z množiny D a pošle ji ostatním procesům
- Shody je dosaženo, pokud se všechny nechybující procesy shodnou na téže hodnot
- Každý nechybující proces Pi nastaví svou rozhodovací proměnnou na d a změní svůj stav na rozhodnutý
- Algoritmus probíhá ve více kolech
- Jedno kolo zahrnuje
    - Poslání zprávy do skupiny procesů
    - Příjem zpráv z předchozího kola
    - Lokální zpracování (rozhodnutí, zastavení)
- Jednoduché neefektivní v pomalých sítích
- Souhlas (dohoda) – výběř hodnoty je shodný pro všechny korektní procesy

45. Popište průběh výměny zpráv v případě 4 Byzantinských generálů (A, B, C, D), z nichž jeden je zrádce (algoritmus shody na vektoru hodnot s poruchou). Nastavované hodnoty jsou následující: A(1), B(2), C(3), D(4).

- Každý generál posílá svoji informaci ostatním

- Po odvysílání všech generálů si uzly mezi sebou vymění vektory s přijatými hodnotami a můžou tak odhalit zrádce

- S využitím hlasovacího mechanizmu může každý generál získat správné hodnoty (opravdu?)

- Problém nemůže být řešen pokud chybný proces lže konstantně

46. Co to jsou inkarnační čísla a kde se používají.

- Čísla identifikující etapu běhu
- Používají se při obnově systému
- Když se systém probouzí, zvětší inkarnační číslo, každá zpráva obsahuje toto číslo a jednotlivé části systému si pamatují aktuální číslo etapy, pokud je inkarnační číslo:
    - > zpráva se zahazuje
    - < čeká se na zprávu obnovy
    - = zpracuje se zpráva

47. Popište algoritmus hlasování (Maekawa 1985).

Proces se snaží získat hlasy ostatních, kdo má nejvíce, může do kritické sekce. Proce může v jeden okamžik hlasovat jen pro jednoho.

Optimalizace komunikační složitost pomocí volebních okresků, pro vstup do kritické sekce je potřeba získat všechny hlasy z vlastního okresku

- Každé dna okresky mají společného člena, velikost okresků je konstantní, každý proces ve stejném počtu okresků

Komunikační složitos odpovídá velikosti okresků.

Prevence deadlocku – logické hodiny (proces ruší původní hlasování, pokud ještě, pak dostane žádost s nižší TM)

48. Popište algoritmus hlasování použitý pro uzamykání kopií a dovolující provádět sdílené čtení a výlučný zápis.

49. Migrace. Co všechno může migrovat a jak. Migrace programu, procesu, paměti, periférií.

Migrace procesu – zlepšení výkonu celého systému

Migrace kódu (slabá mobilita) = přesun kódu ze serveru ke klientovi – vyplnění formulářů, redukuje komunikaci, nepotřebuje spojení, kód lze přesunout na klienta předem. Posílá části klientské aplikace na server místo dat ze serveru na klienta. Zlepšený paralelizmus – webové vyhledávání založené na agentech.

Mechanizmus mobility – slabá mobilita – je spuštěn nový proces, který začíná v novém stavu => daleko jednodušší

Mechanizmus mobility – silná mobilita – migrace procesu

- Klonování procesu,je třeba:
    - Vyjmout proces ze zdrojového plánovače
    - Zmrazit proces – uchovat stav, obsah registrů, …
    - Alokace stavu procesu v cíli
    - Přesunout stav procesu na cíl
    - Přesunout adresový prostor a paměti
    - Forvardovat čekající zprávy ze zdroje na cíl
    - Vyčistit zdroj

Migrace zdrojů – závisí na propojení procesu a zdroje (bind) – podle identifikátoru – web stránka, FTP server

Migrace virtuální paměti.

- Zmražení a kopírování – pozastaví proces, kopíruje všechny paměťové stránky, vyřeší propojení, startuje proces v novém hostu
- Metoda předběžného kopírování – proce pokračuje v činnosti pokud nejsou stránky překopírovány, pak se zmrazí a kopírují se modifikované stránky
- Líná migrace – proces migruje bez přesunu stránek. Stránky migrují když je to potřeba

Migrace komunikačních kanálů – pokud proces komunikuje přes interprocesovou komunikaci (IPC), musí také komunikace migrovat – např. informovat všechny strany spojené s komunikací o novém umístění

50. Rozdíl mezi přemistitelným programem a programem přemistitelným za chodu.

Jako silná / slaba mobilita ? – transparence (přístupu = lokální zdroje jsou přístupné s použitím identických operací,umístění = dovoluje přístup ke zdrojům bez znalosti jejich umístění, migrace,..)

51. Uveďte rozdíl mezi strukturovanými a nestrukturovanými P2P sítěmi.

Strukturované: Chord,Pastry,CAN,BitTorrent

- Soubory jsou přiřazeny specifickým uzlům podle přísných pravidel, lze zajistit efektivní prohledávání a garantovat nalezení souboru, postrádá částečná jména a vyhledávání podle klíčových slov

Nestrukturované: Napster,Gnutella,Kazaa(hybrid – gnutelly a napsteru+ super peer = vystupují jako lokální centra vyhledávání),Freenet(data přenášená v opačném směru než dotaz – nelze zjistit jestli uzel je iniciátorem nebo data jen přenáší = anonymita)

- Nestrukturované decentralizované é soubory mohou být kdekoliv, podporuje částečná jména a dotazování podle klíče, neefektivní prohledávání, negarantuje nalezení informace (gnutella)

52. Co jsou to distribuované hashovaní tabulky(DHT). Na příkladě uveďte, jak fungují (přidání/odstranění dokumentu, přidání/odstranění uzlu).

Distribuovaná verze datové struktury hashovaní tabulka. Ukládá pár (klíč, hodnota). Klíč odpovídá jménu souboru, hodnota odpovídá obsahu souboru.

Cíl: vytvořit efektivní operace pro vkládání, vyhledávání a rušení položek (klíč,hodnota). Každý člen ukládá u sebe podmnožinu párů (klíč, hodnota). Základní operace: nalezení uzlu, který odpovídá klíči.

Mapování klíčů na uzly. Efektivně směrovat požadavky vložení, prohledávání rušení do uzlů.

- Vlastnosti – rovnoměrné mapování klíčů do všech uzlů sítě, každý uzel udržuje informaci pouze o malém počtu uzlů, efektivní směrování zpráv do uzlů
- Připojení a odpojení uzlů ovlivní pouze malý počet uzlů

53. Chord, CAN.

Strukturované P2P sítě, směrovací protokoly DHT

Chord. Uzly jsou organizovány do kruhu identifikátorů. Klíče jsou přiřazeny číselně následujícím uzlům. Hashovací funkce zajišťuje distribuci uzlů i klíčů v kruhu.

Vlastnosti – v systému s N uzly a K klíči s vysokou pravděpodobností – každý uzel obsahuje nanejvýš K/N klíčů, každý uzel udržuje informaci o přibližně O(logN)jiných uzlech, počet prohlížených uzlů závisí na O(logN); -není zaručeno dodání výsledků – není zaručena konzistentnost replik – uzly v kruhu mohou být umístěny v síti kdekoliv (umístění bez vazby na síť)

CAN (Content-Addreeable Network) – založeno na d-dimenzionálním kartézském systému – každý uzle vlastní odlišnou zónu v prostoru – každý klíč je hashován na bod prostoru

54. Jak se od sebe liší jednotlivé stupně pro klasifikaci bezpečnosti operačních systémů?

A – verifikovaná ochrana – vyžaduje úplný formální návrh systému, která je orientován na klasifikaci informace

B – nařízená nebo vynucená ochrana

C – volná ochrana (ponechána na uvážení)

D – bez zajištění bezpečnosti, minimální ochrana MS-DOS

55. Příklady pasivního a aktivního napadení systému a obrana proti tomuto napadení.

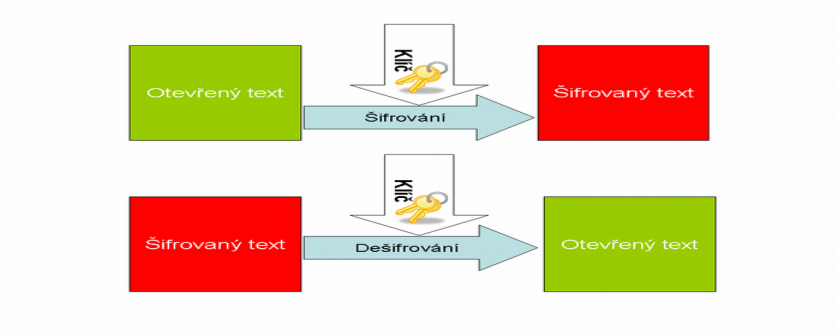
- Pasivní
  - Odposlech, analýza přenosu –odkud,kam, kolik,…
- Aktivní
  - Modifikace, zadržování nebo podstrkávání zpráv
  - Modifikace toku dat – změna obsahu, opakování, změna pořadí, rušení, syntéza zpráv, změna adresy, změna dat, atd…

Obrana

- šifrování – symetrické (konvenční,tajný klíč,jeden klíč)/asymetrické (tajný a veřejný klíč)
- digitální podpisy
- řízení přístupu
- integrita dat
- ověření výměny dat
- vyplňování přenosu
- řízené směrování
- ověřování třetí stranou

56. Uveďte, kde se používá symetrická šifra a kde (k čemu) asymetrická šifra.

Symetrická šifra



- K šifrování i dešifrování se používá jeden klíč

- Symetrické šifry se často používají s asymetrickými. Obvyklé použití je takové, že otevřený text se zašifruje symetrickou šifrou s náhodně vygenerovaným klíčem. Tento symetrický klíč se zašifruje veřejným klíčem asymetrické šifry, takže dešifrovat data může pouze majitel tajného klíče dané asymetrické šifry.
- Asymetrická šifra – vhodné pro bezpečnou komunikaci v prostředích typu internet
- Obé k přímému vzájemnému ověření nebo v protokolech pro nepřímé ověření

57. Problematika výměny klíče. Proč se zavádí tzv. relační klíče?

Relační klíče mají krátkou dobu života – eliminace prozrazení tajného klíče.

IKE – 2 fáze

(a) Vytvoření bezpečného komunikačního kanálu – dohadování kryptografických algoritmů, výměna náhodných čísle – probíhá bez zabezpečení, vytvoření prvního CHILD-SA, který je chráněn klíči odvozenými v této první části

(b) Vytvoření nové CHILD-SA – může být vyvolán jakoukoliv stranou – když útočník zaznamená všechna data poslaná pře bezpečné spojení- nedokáže rekonstruovat klíče pro výměnu mezi CHILD-SA

58. Protokoly pro přímé vzájemné ověření.

59. Protokoly pro nepřímé vzájemné ověření.

Protokoly vzájemného ověření – základní vlastnost – důvěryhodnost, časová souslednost

Přímé – vzájemné ověření dvou entit

- Symetrické šifrování

  A→B: [A||Na]

  B→A: [Nb||f(K||Na)]

  A→B: [f(K||Nb)]

- Asymetrické šifrování

  A→B: Ekb+[A||Na]

  B→A: Eka+[Na||Nb]

  A→B: Ekb+[Nb]

Nepřímé s ověřovacím serverem –protokoly:

- Needham-shroeder
- Otway-ress
- Andrew secure RPC handshake
- Wide-mouthed-frog protokol
- Kerberos protokol

60. Protokoly pro ověření pravosti zpráv využívající symetrické nebo nesymetrické šifrování.

- Ověřování založené na symetrických šifrovacích systémech

Přenos zpráv se děje šifrovaně. Vychází se z předpokladu, že klíč zná pouze vysílač a přijímač informace. Proto se na straně příjemce vysílač ověřuje pouze tím, že se zpráva dešifruje, a pokud dává smysl, chápe se jako vysílač informace jako důvěryhodný. Při šifrování pomocí blokového kódu se postupuje tak, že součástí zprávy je i její zabezpečení. Napadne-li vetřelec zprávu tím, že ji modifikuje, nesouhlasí zabezpečení, a zpráva je odmítnuta. Chyba zabezpečení se může promítnout pouze do napadeného bloku, nebo rozšířit celou zprávou - pak se jeví celá

zpráva jako napadená. Problém v tomto případě je s distribucí tajného klíče. Každá komunikující dvojice musí mít vlastní klíč.

- Ověřování založené na systémech s veřejným klíčem

V tomto případě přijímací strana generuje pár klíčů. Jeden označí jako veřejný a pošle jej vysílací straně. Vysílací strana použije tento klíč k šifrování vysílané zprávy. Přijímací strana zprávu dešifruje tajným klíčem, který si z uvedené dvojice ponechala a tají jej. Protože veřejný klíč může použít každá stanice, která jej zná, je třeba tento protokol doplnit o protokol ověření pravosti vysílací stanice.

61. Hashovací funkce a její použití.

- Nepoužívá klíč (pouze veřejně známý algoritmus)
- Relativně jednoduchý výpočet
- Není reverzibilní (není známa inverzní funkce) – jednosměrná fce
- Obtížné najít pár (x,y) tak, aby H(x) = H(y) – bezkoliznost fce
- Malé změny ve zprávě způsobí náhodné změny výsledku
- Aplikovatelná na blok libovolné (omezené) délky – výstupem je blok pevné délky

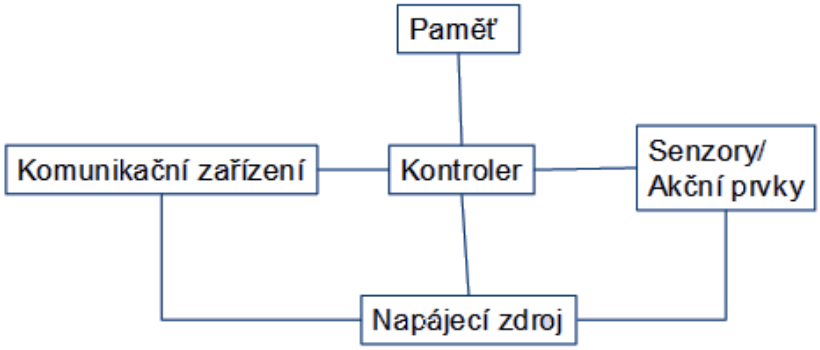
62. Certifikáty a jejich distribuce.

- certifikáty řeší problematiku bezpečné distribuce veřejného klíče
- Vydávání, ověřování a zneplatnění certifikátu řeší certifikační autorita
- Je to blok dat (soubor) obsahující :verzi,sériové číslo, algoritmus podpisu,vystavitele,platnost, předmět,veřejný klíč, distribuční místo, použití klíče, algoritmus miniatury, miniaturu, popisný název
- Získání pomocí prohlížeče – různé autority – prohlížeč se ptá na důvěryhodnost, osobní návštěvou, apod.

63. Požadavky kladené na uzel senzorické sítě. Odlišnosti od „klasických“ sítí.

- musí být levné a malé – počítá se i se zničením
- možnost zpracovávat a ukládat data, bezdrátová vzájemná komunikace
- odolnost proti poruchám
- malé napájecí nároky – procesor s minimální spotřebou schopný přejít do neaktivního stavu
- dostatek paměti pro ukládání dat (s malými napájecími nároky)
- zabudované senzory pro monitorování
- speciální požadavky na bezdrátový přenos – malé vzdálenosti, rychlé přenosy
- požadavek na lokalizaci uzlu (bez využití GPS)
- jiný způsob adresování – nemožnost adresovat jednotlivé uzly, analogie s databázemi – vyhledávání podle klíče
- požadavek na velký počet senzorů v síti – až tisíce (podle potřeby)

64. Architektura uzlu senzorické sítě.



65. Příklady použití senzorických sítí.

- Monitorování stavu budov – seismická aktivita, teplotní roztažitelnost a namáhání (mosty)
- Medicínské aplikace – monitorování zdraví

- Monitorování přírodních dějů – monitorování mořských mikroorganizmů, pohybu zvířat, kontaminace vody, vzduchu...
- Průmyslová automatizace – monitorování teploty, otáček, výšky hladiny, chemického složení...
- Vojenství a bezpečnost
- Převod (neelektrických) veličin na číselnou hodnotu – vlhkost, tlak, náklon, vibrace, rychlost, teplota...

„Bonus“

1) Co jsou to reflektivní DoS útoky?

- Útoky které se snaží zahltit linku oběti, k útoku vždy používaji jiné směrovače, přepínače, počítače
- Můžou být zesílené (smurf) nebo nezesílené
- Smurf
    - vyšle se na broadcast adresu sítě ICMP ECHO REQUES se zfalšovanou adresou odesílatele
    - na broadcast ping odpoví všechny stroje v síti (zesílení N)
- fraggle – obdoba smurf ale s UDP
- TTL záplava – pakety s malou hodnotou TTL, po vypršení je oběť informovaná o vypršení platnosti (zesílaní1,7x)

2) Jaký je rozdíl mezi modelem uploď/download a modelem vzdáleného přístupu v DFS. Uveďte příklad prováděných operací a výhod a nevýhod použití cache

Up/download – přesun souborů

- Čtení souboru = stahování ze serveru
- Zápis souboru – úprava v lokální kopii a následně nahrání souboru na 1
- Výhody = jednoduchost
- Nevýhody – konzistentnost – problém při souběžné modifikaci
    - Neefektivní přenost, pro úpravu určité části 2x přenos celého souboru

Model vzdáleného přístupu

- Souborový systém zajišťuje rozhraní pro provádění standard. Operací (create,delete, read, write, open,close,..)
- Pracuje s bloky souborů – požadavek / odpověď
- Výhody – klient požaduje jen co potřebuje, server může zajistit konzist.
- Nevýhody – mohou být požadovány opakovaně stejná data, může dojít k zahlcení