# Distribuované systémy

štátnicové otázky

# 1. Základní pojmy, modely distribuovaných systémů, obecné vlastnosti.

Dělení systémů:

- SISD – Single Instruction Single Data – jednoprocesorové
- SIMD – Single Instruction Multiple Data – vektorové procesory
- MIMD – Multiple Instruction Multiple Data – paralelní a distribuované
    - paměť sdílená (multiprocesory) / nesdílená (multipočítače), propojení sběrnice / přepínače, zpoždění – těsně vázané (paralelní systémy) / volně vázané (**distribuované systémy**).

**Distribuovaný systém** - soubor nezávislých počítačů, které se jeví svým uživatelům jako jeden souvislý systém.

**Důvody pro zavádění** distribuovaných systémů jsou zvýšení **dostupnosti**, zvýšení **výkonnosti** a zvýšení **spolehlivosti**.

Rozeznáváme **dva základní** modely distribuovaných systémů:

- Model **klient – server** – služby implementovány na serverech, klienti pouze přístup k serverům
- **Integrovaný model** – v každém uzlu přítomen kód pro realizaci požadovaných funkcí

## Předpoklady pro realizaci distribuovaných systémů

- **Chyby** – Uzly v síti musí vykazovat nezávislé chybové režimy. Chyba v jednom uzlu nesmí ohrozit činnost jiného uzlu.
- **Jména** – Zdroje musí být jednoznačně označovány a lokalizovány.
- **Distribuované řízení** – řízení je rozložené na jednotlivých uzlech sítě, neexistuje centrální uzel.
- **Heterogenita** – počítače s různým zobrazení dat, různým instrukčním kódem různou architekturou a různými operačními systémy

## Požadavky na distribuovaný systém

- **Ochrana** –ochranu svých zdrojů před neoprávněným přístupem uživatelů obrana pomocí speciálních prostředků, jako je ověřování uživatele a jeho práv, obranné valy apod.
- **Duplicita** –zvýšením počtu kritických zdrojů (zdvojení tiskáren, disků, atd.) -> zvýšená průchodnost systému.
- **Transakce** – Podpora transakcí dovoluje provádět efektivně souběžnou manipulaci se sdílenými daty.

## Transparentnost v distribuovaných systémech

| Transparentnost | Význam |
|---|---|
| **Přístupu** | Lokální a vzdálené zdroje jsou přístupné s použitím identických operací. |
| **Umístění** | Dovoluje přístup ke zdrojům bez znalosti jejich umístění. |
| **Migrace (mobilita)** | Zdroj může být přesunut v systému na jiné místo, aniž by se tím ovlivnil výpočet. |
| **Výkonnosti** | Dovoluje, aby byl systém rekonfigurován podle toho, jak se mění výkon. |
| **Replikace** | Dovoluje pracovat s více instancemi zdroje bez toho, že by uživatelé věděli, že jde o repliky. Cílem je vyšší spolehlivost a výkonnost. |
| **Souběžnosti** | Dovoluje několika procesům souběžně přistupovat ke sdíleným zdrojům bez interference mezi nimi. |
| **Chyb** | Skrývá chyby zdroje a chyb obnovu po chybách. |
| **Škálovatelnosti** | Dovoluje rozšiřovat systém i aplikace beze změny struktury systému nebo aplikačních algoritmů. |

# 2. Příklady aplikací typu klient-server a peer-to-peer

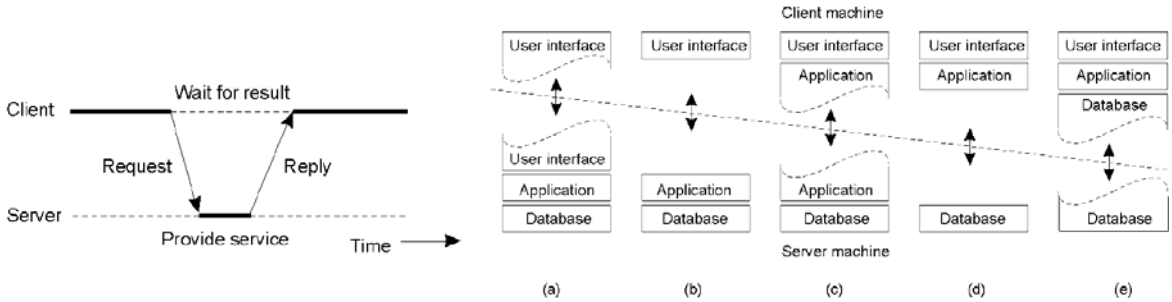
## Model klient-server

Architektura klient/server

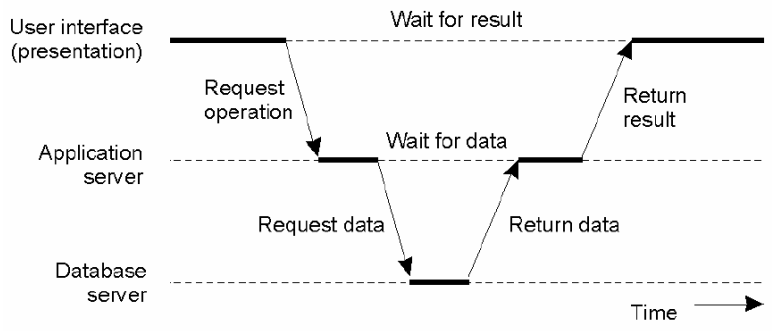
- Server musí být výkonný, spolehlivý, známý zdroj dat
- Klienti požadují data od serverů
- Často používaný model

Omezení architektury:

- Těžko dosažitelná škálovatelnost
- Představuje úzké místo z hlediska chyb
- Vyžaduje administraci
- Mohou existovat nevyužité zdroje




**Obrázek 1 – Obecná a alternativní (různé rozdělení úkonů obou stran) interakce mezi klientem a serverem**




**Obrázek 2 – Příklad, kdy server funguje jako klient**

## Model peer-to-peer

Základní dělení P2P systémů je na systémy **sdílející obsah**, nebo na systémy **vyhledávající obsah**.

**Sdílení obsahu:**

- *Přímý přenos mezi uzly* – identické pro všechny uzly
- Strukturované nebo nestrukturované ukládání dat
- *Automatická replikace dat*

Jiné dělení P2P systémů:

- Pro paralelní výpočty
  - Mnohonásobné výpočty
  - Rozdělené výpočty
- Přístup k souborům a informacím
  - Sdílení obsahu / Přístup k souborům
  - Filtrování, vyhledávání
- Kooperující (RT)
  - Instant messaging, sdílené aplikace, hry

Rozdělení podle decentralizace

- Čisté P2P systémy
  - Členové jsou servery i klienti (servent)
  - Bez centrálního serveru, bez centrálního směrovače
- Hybridní P2P
  - Centrální server pro udržování informací o členech
  - Odpovídání na dotazy (indexy)
  - Členové udržují informaci (soubory)
  - Směrování pomocí převodu reference – adresa
- Kombinované
  - obojí

### *P2P směrování*

- členové jsou uzly, sousední členové jsou propojeni virtuálními hranami (TCP, IP adresa).
- Periodický ping zjišťuje dostupnost jednotlivých uzlů.
- Při přenášení zpráv je prováděn test životaschopnosti.
- Rekonfigurace topologie se provádí při přidávání nového uzlu nebo při ztrátě souseda.

### *Aplikace P2P sítí*

- sílení souborů (Napster, Gnutella, Kazaa)
- síťové hry (DOOM)
- sdílené aplikace (ICQ, sdílené tabule)

### *Napster*

- centrální server -> problém z hlediska škálovatelnosti a chyb
- při hledání je použit seznam od všech uživatelů -> odpověď IP adresa počítače, který soubor vlastní

### *Gnutella*

- decentralizované prohledávání -> dotazování sousedů -> počet úrovní prohledávání je dán TTL

### *Kazaa*

- Hybrid
- Obsahuje tzv. super-peer (obdoba Napster serveru pro malou část sítě, vybíráni na základě parametrů), což jsou uzly vystupující jako lokální centra prohledávání.

### *Freenet*

- Data jsou přenášena v opačném směru než dotaz.
- Není možné pak zjistit, zda je uživatel iniciátorem nebo pouze přenáší data dál nebo je spotřebovává.

# 3. Komunikace mezi procesy, volání vzdálených podprogramů, vzdálené volání metod.

Pro komunikaci se používají dva základní prostředky – programová rozhraní:

- Systémy s posíláním zpráv
- Volání vzdálených podprogramů

## Systém posílání zpráv

1. **Blokované/neblokované** operace (synchronní/asynchronní)
   Při blokované operaci je proces vyvolávající požadavek pozastaven do doby, kdy volaný proces požadavek nezpracuje. Blokovaná operace je obdobou volání vzdálených podprogramů.
   Neblokované operace mají význam tam, kde je třeba, aby volající proces mohl souběžně provádět další operace.
2. **S využitím vyrovnávací paměti / bez využití vyrovnávací paměti**
   Vyrovnávací paměť, obsahující zprávu se může nacházet
   - V paměti odesílatele
   - V paměti komunikačního programového vybavení odesílatele
   - V paměti komunikačního programového vybavení příjemce
   - V paměti příjemce
3. **Spolehlivý / nespolehlivý komunikační protokol**
   spolehlivé - spojově orientované protokoly a spolehlivé diagramové protokoly, větší režie
4. **Pevná / proměnná délka zprávy**
5. **Přímá / nepřímá komunikace**
   Přímou komunikací se rozumí komunikace mezi procesy, tj. cílová i zdrojová adresa je vztažena přímo k procesu.
   Při nepřímé komunikaci se používá pro označení odesílatele i adresáta číslo portu, tj. odkaz na datovou strukturu, která může existovat nezávisle na procesu.
6. **Mapování adres**
   Mapování adres se provádí následujícími způsoby
   - Zvláštním protokolem – příkladem je DNS, dalším příkladem převodu jednoznačného identifikátoru skupiny programů na port je portmapper (RPC).
   - Při překladu – tabulkou dvojic jméno – adresa.
7. **Rendezvous**
   Synchronní přenos zpráv mezi procesy. Nevyužívá vyrovnávací paměti, data jsou přenášena přímo z paměti odesílatele do paměti příjemce.

## RPC (Remote Procedure Call) - Volání vzdálených procedur

- při RPC je volající proces pozastaven, parametry volání uloženy do zprávy, a ta přenesena do vzdáleného uzlu -> zde je provedeno vyvolání požadovaného podprogramu, výsledek volání uložen opět do zprávy, a ta přenesena do uzlu volajícího procesu
- volající proces je po přijetí zprávy a zpracování parametrů aktivován tak, jako by volal lokální proceduru.
- Jednoduchost činí RPC atraktivním jako primární mechanismus pro komunikaci v distribuovaných programech

### *Problémy RPC*

- **spojování (binding)** - jak bude volající označovat volanou proceduru, a jak ji najde
- **heterogenita** - jak systém komunikuje s počítači různých typů a s programy zapsanými v různých programovacích jazycích
- **transparentnost** - jak je třeba upravit sémantiku při volání RPC oproti lokálnímu volání procedury

- **vzájemné působení (concurrency)** - jaký mechanismus je použit pro vzájemné působení, a jak spolupracuje s komunikačním mechanismem.

### *Heterogenita*

- volající a volaný se musí shodovat v typech argumentů a výsledku procedury -> správná deklarace rozhraní
- automaticky **generované spojky (stub)** – konverze mezi různými formáty apod. -> při vyvolání vzdálené procedury se nejprve vyvolá lokální spojka klienta pro tuto proceduru. -> Spojka klienta uloží odpovídající reprezentaci argumentů do zprávy a inicializuje výměnu zpráv s počítačem serveru. Spojka obsahuje všechny detaily konverze a komunikace.

Pro reprezentaci dat se používají tzv. specifikační jazyky, které dovolují kromě základních datových typů obecně popsat i další, uživatelem zavedené strukturované typy. Mezi nejčastěji používané patří:

- XDR (eXtended Data Representation) Implicitní vyjádření typu dat, používá se u SUN RPC
- ASN.1 Explicitní vyjádření typu, používá se např. v SNMP

omezení RPC oproti "normálnímu" podprogramu:

- parametry se přenáší hodnotou
- v podprogramech nemohou být globální proměnné
- jako parametry nelze přenášet odkazy na funkce nebo procedury

### *Sémantika volání*

- **funkci vyvoláme pouze jednou** - Nastane-li chyba, chybu nahlásíme nebo vyvoláme správce výjimek, který zjistí, ve které fázi chyba nastala a sjedná nápravu.
- **neomezené čekání** - Po vyvolání funkce budeme neomezeně dlouho čekat na její ukončení. To vede k zablokování procesu.
- **opakované volání** - Po uplynutí nastaveného časového úseku se volání opakuje. To může ale vézt k chybě. Pouze u tzv. idempotentních operací (vícenásobné opakování neovlivní stav systému) nedojde k chybě.

Obecně rozeznáváme následující sémantiky:

- **provedení operace nejvýše jednou** - Funkce se vyvolá právě jednou. Po vypršení nastaveného časového kvanta se ohlásí chyba. Operace nemusí být ve skutečnosti provedena ani jednou (chyba při vyslání zprávy), nebo právě jednou (chyba po provedení operace), nebo započata a nedokončena (chyba při provedení).
- **provedení operace alespoň jednou** - Klient vyvolá funkci a očekává její výsledek (potvrzení). Nedostane-li do určité doby odpověď, opakuje volání funkce. Požadovaná operace může být provedena jednou, ale i vícekrát. Je-li požadovaná operace idempotentní, odpovídá sémantika lokálnímu volání.
- **provedení operace právě jednou** - Tato sémantika vyžaduje použít spolehlivé komunikace, aby se vyloučila zbytková komunikační chyba. V případě výpadku serveru se jedná o závažnou chybu.
- **získání výsledku poslední prováděné operace** - Tato sémantika se blíží sémantice lokálního volání. Jako výsledek volání se chápe výsledek posledně prováděné operace.
- **provedení operace jednou nebo vůbec** - Tato sémantika předpokládá, že k provedení operace ve vzdáleném uzlu je využit transakční mechanizmus.

## RMI (Remote Method Invocation)

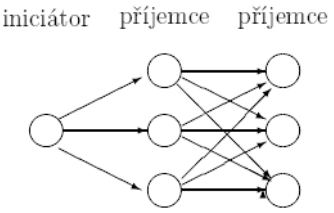
- volání metod nad vzdáleným objektem transparentně
- komunikace přes síťovou vrstvou, která je vytvořena podporovaným programovým jazykem (Java)
- síťová vrstva zajišťuje úplnou transparentnost (výjimky, předávání odkazem apod.)

# 4. Spolehlivé protokoly pro vysílání (broadcast)

- posílání požadavků / zpráv celé skupině najednou
- počet členů skupiny a jejich složení se mění
- členové skupiny musí mít možnost komunikovat navzájem
- zajištění převední zprávy 1:N na N zpráv 1:1

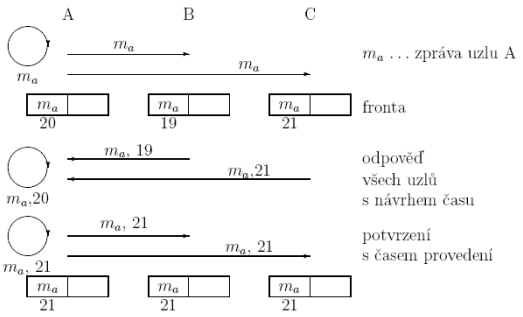
## Atomický broadcast protokol

- Uzel (iniciátor) pošle zprávu všem členům skupiny
- Uzel po prvním obdržení zprávy pošle stejnou zprávu ostatním
- Po obdržení N-1 zpráv je zpráva provedena
- Nevýhoda: velké zatížení, vše se musí provést v určitém časovém okamžiku, nutnost pamatovat si všechny zprávy do doby, než nedostanu kopie od ostatních




### *Protocol ABCAST*

- Iniciátor pošle zprávu s kódem operace všem členům skupiny
- Všechny uzly odpoví s časovou značkou = návrh času na provedení operace
- Iniciátor vybere nejvyšší hodnotu a odešle všem uzlům
- Jednotlivé uzly obsahují frontu požadavků, které se mají provést (před provedením musí být potvrzeny všemi uzly) -> ve všech uzlech se provedou operace ve stejném pořadí, ale ne ve stejné době




### *Modifikace ABCAST protokolu*

- kruhová topologie – iniciátor pošle zprávu všem členů skupiny, ty přiloží ke zprávě časovou značku, po vrácení k iniciátorovi pošle iniciátor v druhé fázi zprávu s časem provedení -> složitost 2 oběhy
- stromová topologie – iniciátor kořenem stromu -> souběžné vysílání po hranách stromu
- další možnosti: sběrnicová topologie, přechod od potvrzování pozitivního k negativnímu (posílá se pouze nesouhlas s vlastním navrhovaným časem)

## CBCast (protokol s oslabeným uspořádáním)

Požadavky:

1. Zprávy, generované nezávislými procesy mohou do cílového procesu dorazit v různém pořadí.
2. Zprávy, generované jedním procesem mohou do cílového uzlu dorazit v různém pořadí, ale cílový proces je musí zpracovat v původním pořadí.
3. Existuje-li mezi procesy závislost a → b a b → c, pak je tato závislost tranzitivní (a → c).

**Řešení:** zpráva přenáší svoji historii (např. čísla předchozích zpráv)

## GBCast (broadcast protokol pro dynamicky se měnící skupiny)

nadstavbový protokol, zajišťuje dynamický vznik a zánik skupin

1. Každý uzel obsahuje tabulku se jmény existujících skupin a procesů ve skupinách.
2. Při inicializaci musí uzel obdržet výše uvedenou tabulku. Tu opraví a vyšle ji všem uzlům, Opustí-li proces skupinu, musí se o tom dovědět všichni ostatní členové skupiny.
3. Provádí-li skupina operaci, musí být použity zámky.

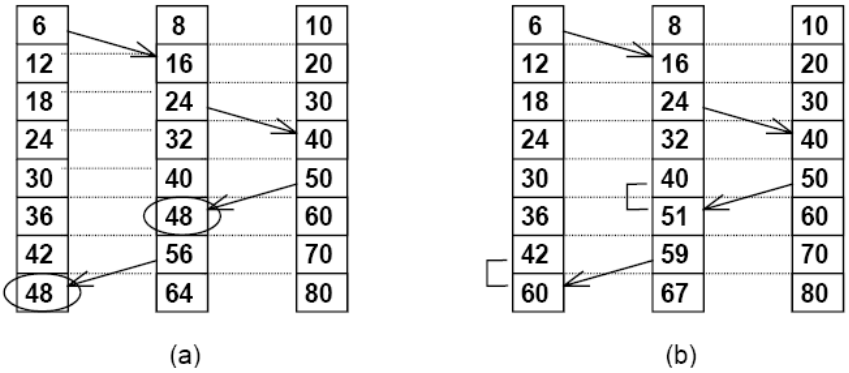
# 5. Čas, synchronizace času, logické a fyzické hodiny, algoritmy synchronizace, časové servery.

obecné vlastnosti distribuovaných algoritmů:

- informace potřebná pro rozhodování je rozprostřena mezi několika uzly
- procesy se rozhodují na základě lokálních informací
- měly by být vyloučeny takové komponenty, jejichž selhání způsobí havárii systému
- neexistují společné hodiny ani přesné měření globálního času

## Uspořádání událostí a logické hodiny (Lamportův algoritmus)

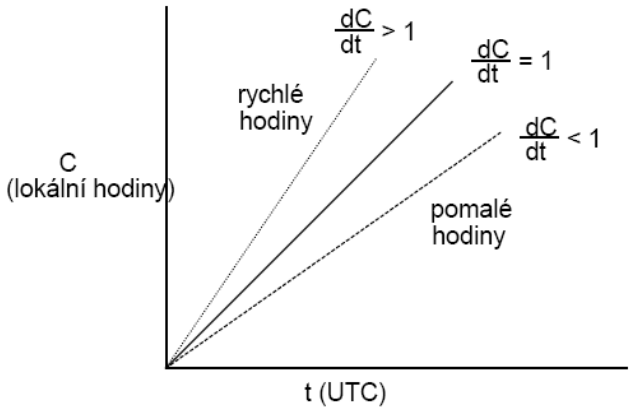
- Logické hodiny zajišťují konzistentní měření času v celém systému, nezajišťují však žádným způsobem měření reálného času.
- Důležité je aby se procesy shodly na pořadí v jakém se staly jednotlivé události
- C(a) – čas, ve kterém se stala událost *a* a na kterém se shodly všechny procesy
- jestliže a → b, pak C(a) < C(b).
- $T_m$ časová značka zprávy *m*
  - Proces *i* vysílá v čase $C_i(a)$ zprávu *m*; $T_m = C_i(a)$
  - Proces *j* přijme zprávu *m* v čase $C_j(b)$. Pak $C_j = \max(C_j(b), T_m+1)$




**Korekce hodin dle Lamportova algoritmu**

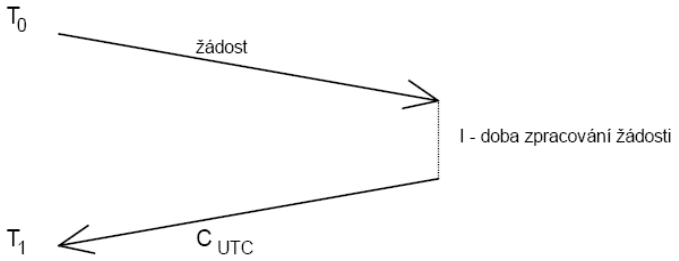
## Synchronizace fyzických hodin

- Zavedení externích fyzických zdrojů času




**Rozdílné rychlosti lokálních hodin**

### *Cristianův algoritmus*



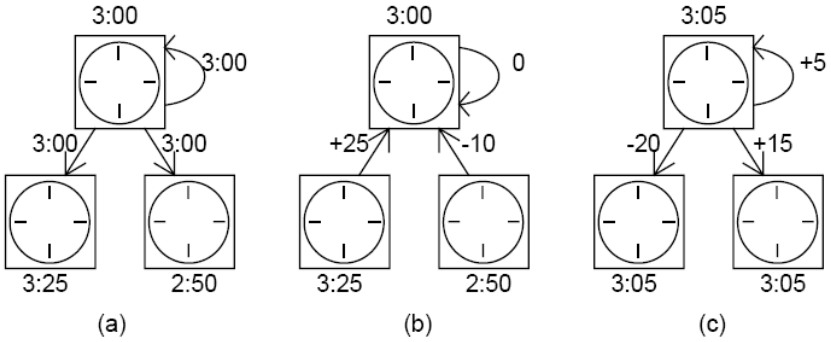

o

- jeden pasivní time server
- k synchronizaci dochází periodicky
- $T = T_{UTC} + (T_1 - T_0 - I)/2$

### *Berkeley algoritmus*

(Berkeley UNIX)

- aktivní time server
- periodicky se ptá ostatních na rozdíl času, počítá průměr, vrátí rozdíl




**Obrázek 3 - (a) Časový server zahájí synchronizaci hodin (b) Odpovědi obsahující časový rozdíl (c) Zprávy s určením opravy lokálních hodin**

## Časový server

- slouží k přenosu informace o přesném čase
- problém: při přenosu informace mezi dvěma počítači dochází ke zpoždění, které není konstantní
- jednoduché řešení: zpoždění se nebere v úvahu, je vrácen „přibližný“ čas
- ke zvýšení přesnosti nastavení času se využívá spolupráce několika nezávislých časových serverů, které získávají informaci o přesném čase od primárních časových zdrojů (časových etalonů) prostřednictvím přímého radiového spojení, u kterého lze předpokládat takřka konstantní zpoždění šíření signálu

# 6. Transakce, jejich realizace, zajištění konzistentnosti, vnořené transakce, distribuované transakce.

**Transakce** – provádění činností, které nelze (nebo nechceme) od sebe oddělit, po odsouhlasení uzavření transakce od všech účastníků transakci provedeme, jinak se provedené změny neprojeví

## Vlastnosti transakcí

- **Atomicita** - pro vnější svět je transakce nedělitelná, provede se buď celá nebo vůbec
- **Konzistence** - transakce neporušuje vnitřní konzistenci
- **Izolovanost** (sekvenčnost) - konkurentní transakce se navzájem neovlivňují
- **Trvanlivost** - po úspěšném ukončení transakce jsou změny trvalé

## Vnořené transakce

Uvnitř transakce mohou být volány další transakce -> při neprovedení vnější musí být zrušeny i vnitřní -> nutná režie

## Zotavení z chyb

1. **Zpětné zotavení -** Návrat do posledního konzistentního stavu, tj. všechny změny vyvolané transakcí se zruší.
2. **Dopředné zotavení** - Nastavení výsledného konzistentního stavu, tj. v případě dostatečného množství informací následné dokončení transakce.

## Transakční komunikační primitiva

| | | |
|---|---|---|
| T_begin | M→S | Začátek transakce |
| T_end | M→S | Explicitní konec transakce |
| T_ready | M←S | Připravenost na commit |
| T_commit | M↔S | Commit - potvrzení transakce |
| T_abort | M↔S | Zrušení transakce |

### *Dvoufázový commit (potvrzování)*

- Pro zabezpečení atomicity uzavření transakcí
- Master posílá všem ostatním požadavek o ukončení -> každý slave po obdžení zjistí zda je transakci schopen uzavřít
    - Pokud ano, uzamkne všechny objekty a vrátí potvrzení
    - Pokud ne, pošle zprávu o zrušení transakce
- Jestliže master obdržel od všech kladné potvrzení, vyšle všem slave uzlům commit
- Uzly provedou ukončení a uvolní objekty -> pošlou zpět potvrzení o ukončení
- Poté co master obdrží všechna potvrzení je transakce ukončena

## Obnova transakcí po chybě

- **seznam požadavků** – co se má udělat po chybě
- **stínové stránky** – kopie originálu, všechny změny se provádí na kopii, je-li vše v pořádku, přehodí se odkaz z kopie na originál
- **záznam do souboru logů** – ukládání informací potřebných k „vycouvání“ nebo dokončení transakce

## Souběžné provádění transakcí

### *Zámky*

- Objekt se při provádění transakce uzamkne, čímž je k němu znemožněn ostatním procesům přístup -> příliš omezující

- Rozdělení zámků: pro čtení a pro zápis
  - "many readers / single writer" (více čtenářů / jeden zapisovatel).

| | Read | Write |
|---|---|---|
| Read | OK | - |
| Write | - | - |

- **granularita zámků –** zámky vztaženy buď na celý soubor nebo databázi (hrubá granulace) anebo na jeden záznam či blok souboru (jemná granulace) -> větší režie, větší stupěň paralelismu, snadněji dojde k deadlocku (uvíznutí)

### *Dvoufázové uzamykání*

- v první fázi všechny požadované objekty uzamčeny, poté jsou na nich provedeny požadované operace
- v druhé fázi, jsou všechny zámky odstraněny, provádí se až při ukončení transakce, ať již úspěšném nebo neúspěšném

### *Optimistická kontrola konkurence*

- každá transakce provádí cokoliv bez ohledu na jakékoliv jiné transakce
- pouze v případě, že dojde ke konfliktu, je transakce abortována.
- Implementace pomocí lokálního pracovního prostoru -> při commitu kontrola, zda k některým objektům nepřistupovala zároveň jiná transakce -> potvrzení nebo zrušení transakce
- nedochází k deadlockům
- nevýhoda: při velkém zatížení mnoho konfliktů -> mnoho transakcí se provádí několikrát

### *Časové značky*

každá transakce při startu dostane svoji časovou značku T.

Každému souboru jsou přiřazeny dvě časové značky TR a TW, které určují transakci (resp. její časovou značku), která ze souboru naposledy četla, resp. do něj zapisovala.

| | T < TR | T < TW |
|---|---|---|
| čtení | OK, novější transakce četla správná data | abort, data jsou přepsána novější transakcí |
| zápis | abort, novější transakce četla stará data | OK, ale data nezapisovat, novější transakce už zapsala nová data |

## Ukončování transakcí

- **postupné** uvolňování zámků se děje v pořadí, ve kterém byly přidělovány.
  - nevýhoda: dojde-li k chybě v transakci po uvolnění některých zámků, musí se opravit i ty transakce, které mezitím opravenou hodnotu požily
- **souběžné** uvolnění zámků odstraňuje předchozí nedostatky - zámky jsou uvolněny teprve v okamžiku, kdy je již jasné, jak provedení transakce dopadne

## Distribuované transakce

Jsou prováděny na více uzlech -> problém: ukončení může nastat v lib. uzlu

### *Metoda dvoufázového ukončování*

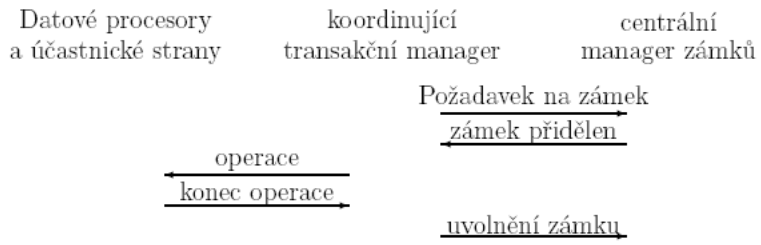
musí být zvolen koordinační server, ostatní jako cohorts (sluhové), protože musí poslouchat koordinátora

- označí transakci jako podmíněně spáchanou
- pošle všem sluhům zprávu připraveni na commit?
- pokud jsou sluhové před abortem, nebo již transakci zrušili, pošlou negativní odpověď
- v opačném případě se sluhové připraví na commit zapsáním seznamu změn do logového souboru. Stav lokální transakce je označen jako podmíněné spáchání, transakce není dosud spáchána. Sluha pak pošle koordinátoru kladnou odpověď.
- jestliže koordinátor obdrží třeba i jedinou negativní odpověď, zruší příslušnou transakci a všem sluhům posílá zprávu abort. Sluhové pak ruší vlastní transakce.

- při obdržení kladných odpovědí ode všech sluhů, označí koordinátor stav transakce commited (spáchána) a posílá všem sluhům zprávu pokračuj a spáchej
- každý sluha po přijetí této zprávy označí transakci jako spáchanou a ukončí ji jako v případě jednoho serveru.

Tři skupiny ukončování

- centralizované – koordinátor posílá všem
- lineární – koordinátor posílá prvnímu uzlu, ten dalšímu atd…
- distribuované – uzly si posílají navzájem




**Centralizované dvoufázové ukončení transakce**

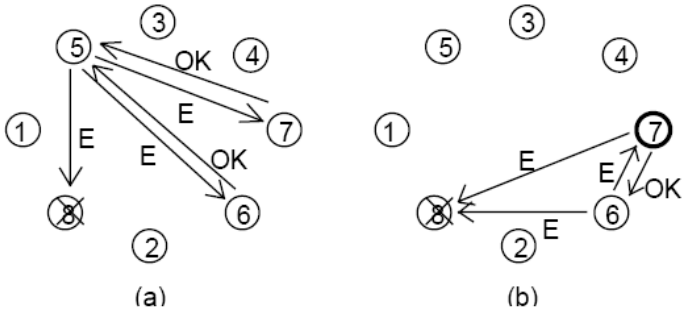
# 7. Distribuované algoritmy, algoritmy výběru, vzájemného vyloučení, shody a ukončení

## Algoritmy výběru 1 z N

- úplný graf – Bully algoritmus
- kruhová topologie
  - logický kruh – předávání dle seznamu, rekonstrukce kruhu
  - fyzický kruh – předávání dle sousedství uzlů
- stromová topologie – logický strom

### *Bully algoritmus*

- Když se proces rozhodne volit, zašle zprávu všem procesům s vyšší identifikací (číslo procesu apod.)
- Když přijde odpověď, proces končí, když nepřijde nic, proces vyhrál, je novým koordinátorem a pošle o tom zprávu všem ostatním




**Bully algoritmus**

## Algoritmy vzájemného vyloučení

- Základní rozdělení
  - Centralizované metody (sequencer) – řízeno centrálním prvkem
  - Decentralizované metody
    - Založené na soupeření
    - Založené na předávání pověření
- Decentralizované metody založené na soupeření
  - Lamportův algoritmus – libovolná topologie, časové značky (3 fáze)
  - Ricart a Agrawalla – libovolná topologie, časové značky (2 fáze)
  - Maekawa – vytváření hlasovacího quora
- Decentralizované metody založené na předávání pověření
  - Suzuki a Kasami – broadcast
  - LeLann – logický kruh
  - Reymond – stromová struktura, rozšíření pro sdílení K identických zdrojů

### *Lamportův algoritmus*

- Proces vyšle žádost o přístup do kritické sekce a čeká, až dorazí odpovědi od všech procesů, pokud všechny žádosti v jeho frontě mají větší časovou značku, může vstoupit do KS

### *Ricard & Agrawala algoritmus*

- Když chce proces vstoupit do kritické sekce, zašle žádost s časovou značkou ostatním procesům a čeká na všechny došlé odpovědi s potvrzením možnosti vstupu.

Když proces přijme zprávu se žádostí, pak:

1. Jestliže příjemce není v kritické sekci a ani do ní nechce vstoupit, pak pošle zpět zprávu s potvrzením.
2. Jestliže je příjemce v kritické sekci, pak neodpovídá, požadavek si zařadí do fronty.

3. Jestliže příjemce ještě není v kritické sekci, avšak chce do ní vstoupit, porovná časovou značku přijaté žádosti s časovou značkou vlastní žádosti. Pokud vlastní žádost má nižší časovou značku, tj. byla vyslána dříve, pak neodpovídá a zařadí žádost odesílatele do fronty. V opačném případě, kdy byla žádost odesílatele odeslána dříve, pošle příjemce zpět zprávu s potvrzením.

## Algoritmy shody

**Shoda** - N procesů se chtějí dohodnout na hodnotě

### *Obecný postup*

- Všechny procesy $P_i$ začínají ve stavu „nerozhodnutý“
- Každý $P_i$ navrhne hodnotu $V_i$ z množiny $D$ a pošle ji ostatním procesům
- Shody je dosaženo pokud se všechny nechybující procesy shodnou na téže hodnotě $d$ Každý nechybující proces $P_i$ nastaví svou rozhodovací proměnnou na $d$ a změní svůj stav na “rozhodnutý“
- Algoritmus probíhá ve více kolech (round).

### *Byzantinští generálové*

- Tři nebo více generálů se potřebuje vzájemně seznámit se svými záměry
- Jeden nebo více generálů může být zrádce, který dává chybné informace
- K řešení tohoto problému používají protokol
  - Každý z generálů posílá svou informaci ostatním (předpokládáme spolehlivou komunikaci)
  - Jakmile generál shromáždí všechny hodnoty, pošle vektor hodnot ostatním generálům
  - S využitím hlasovacího mechanizmu může každý generál získat správné hodnoty
- Problém není řešitelný pro $N \leq 3$
- Pro k zrádců je třeba 2k+1 loajálních generálů
- Problém nemůže být řešen pokud chybný proces lže konzistentně
- Lze použít zabezpečení zprávy (šifrování, digitální podpis)

## Algoritmy ukončení a detekce globálního stavu

- Algoritmus synchronizace ukončení má za úkol zabezpečit následující: V případě, že všechny procesy jsou ve stavu *ukončen*, pak se v konečném čase tuto skutečnost všechny procesy dozví.

### *Dijkstra-Scholten (DS) algoritmus*

- Jestliže graf procesů je strom, pak každý listový proces při přechodu do stavu *ukončen* pošle signál svému otci
- Jakmile proces dostane signály od všech svých synů, pošle signál svému otci
- Jestliže všechny signály dostane iniciační proces, je distribuovaný výpočet ukončen.

### *Detekce globálního stavu (Chandy-Lamport)*

- Iniciátor vyšle všem výstupním uzlům značku.
- Příjem první značky:
  - Uzel si zapamatuje poslední přijaté a odeslané zprávy = stav uzlu
  - Stav všech příchozích kanálů označí za prázdný
  - Vyšle všem výstupním uzlům značku
- Příjem zpráv od uzlů, od kterých ještě nepřišla značka:
  - Zapamatuje si čísla zpráv
- Příjem značek od dalšího uzlu:
  - Stav kanálu = zaznamenané zprávy došlé od uzlu mezi přijmutím první značky a značky od tohoto uzlu
- Algoritmus končí po přijmutí všech značek. Zaznamenané stavy uzlů a kanálů definují konzistentní stav systému.

# 8. Uvíznutí (deadlock) v distribuovaných systémech

## Distribuovaný deadlock

- Soubor procesů, blokovaných čekáním na podmínky, které nemohou nastat
- Starvation – procesu je bráněno v pokračování (vyhladovění)
    - není deadlock

## Podmínky vzniku deadlocku

- stálé zdroje
    - vzájemné vyloučení
    - postupné předělování
    - nepreemptivní plánování
    - neomezené čekání
- dočasné zdroje
    - nehierarchické volání
    - neomezená doba odezvy

## Zpracování deadlocku

- prevence (prevention) – principiálně nepřipustím deadlock
- zamezení (avoidance) – vyhnu se deadlocku včasným testováním
- detekce a odstranění (detection and resolution)
    - centralizovaný algoritmus
    - decentralizovaný algoritmus
    - hierarchický algoritmus

## Prevence deadlocku

- např. hierarchické přidělování zdrojů
    - každý zdroj má přiděleno ohodnocení
    - o zdroje se žádá vzestupně
    - uvolňují se sestupně
    - při znovupřidělení zdroje je třeba všechny zdroje ze skupiny uvolnit a pak znovu přidělit

## Zamezení deadlocku

- uspořádání požadavků, nemusí být efektivní
- všechny požadavky se musí znát dopředu
- zdroje musí být předem dány, požadavky na ně také
- pro každý požadavek se musí provést analýza stavu
- např. bankéřův algoritmus
    - je dána jistina bankéře (kolik může max. půjčit)
    - jsou dány max. požadavky klientů
    - podmínky úlohy – jinak nemá ceny řešit
        - žádný z požadavků nesmí být větší než jistina
        - součet požadavků musí být větší než jistina
    - bankéř je ve stavu jistý, testuje vznik stavu nejistý
    - při požadavku na další zdroje "dopočítává" vývoj půjčování pro nejhorší případ

## Detekce a odstranění deadlocku

- detekce deadlocku
    - udržování grafu dostupnosti zdrojů, hledání deadlocku
- odstranění deadlocku
    - přerušení zacyklení grafu, přerušení jednoho nebo více procesů a vrácení jejich zdrojů
    - procesy se mohou vracet k synchronizačnímu bodu

## Detekce distribuovaného deadlocku

- **centralizovaný algoritmus**
  - koordinátor určuje globální WFG (Wait For Graph) a hledá cykly
    - je jednoduchý
    - Ho a Ramamoorthy, jedno a dvoufázový algoritmus
- **distribuovaný algoritmus**
  - globální WFG se schopností detekce rozšířené na více uzlů
    - Obermark algoritmus path-pushing (strkáním cesty)
    - Candy, Mistra, Haas – edge-chasing (vytepávání hran)
    - difuzní
- **hierarchický algoritmus**
  - hierarchická organizace, strany detekují deadlock pouze zahrnutím svých potomků
    - Menasce a Muntz algoritmus
    - Ho a Ramamoorthy algoritmus

### *Jednoduchá a centralizovaná detekce deadlocku*

- všichni přidělují a uvolňují zdroje posíláním zpráv koordinátorovi
- koordinátor udržuje graf a detekuje deadlock
- problémy
  - zatížení komunikačních linek
  - úzké místo systému
  - může také detekovat "zdánlivý deadlock"

### *Ho a Ramamoorthy dvoufázový centralizovaný algoritmus detekce deadlocku*

- každá strana si udržuje 2 tabulky
  - tabulku přidělených zdrojů (zdroj přidělen procesu)
  - tabulku čekajících procesů (proces čeká na zdroj)
- koordinátor periodicky požaduje zaslání všech stavových tabulek, vytváří WFG a testuje deadlock
  - nejsou-li cykly, není deadlock
  - nalezne-li cyklus, vyžádá si opět tabulky a znovu buduje WFG
  - používá ale pouze ty hrany, společné oběma souborům stavových tabulek
- důvodem je to, že pokud použije informaci ze dvou po sobě jdoucích záznamů, získá koordinátor konzistentní pohled na stav.
  - algoritmus redukuje možnost indikace falešných deadlocků, ale úplně je neodstraňuje
- nalezení cyklu v tabulce neznamená existenci dreadlocku

### *Ho a Ramamoorthy jednofázový centralizovaný algoritmus detekce deadlocku*

- každá strana si udržuje dvě tabulky
  - všechny lokální procesy a zdroje, které jsou uzamčené
  - zdroje uzamčené v této straně od lokálních i nelokálních procesů
- strana periodicky požaduje obě tabulky, konstruje graf WFG
- WFG obsahuje pouze informaci o nelokálních procesech, jestliže tato informace je shodná pro procesy strany i zdroje strany
  - detekuje-li se cyklus, jde o deadlock,
  - nedetekuje-li se, nejde o deadlock
- detekuje korektně deadlocky eliminujích nekonzistentnost podávání zpráv vlivem zpoždění
- vyžaduje více prostoru než 2-fázové HR

## Distribuovaná a hierarchická detekce a odstranění deadlocku

Detekce

- **distribuovaný algoritmus**
  - Obermarckův path-pushing algoritmus
  - Chandy, Misra a Haas edge-chasing

- **hierarchický algoritmus**
  - Menasce a Muntz algoritmus
  - Ho a Ramamoorthy algoritmus

## Distribuovaná detekce deadlocku

**path-pushing**

- WFG je rozdělen na cesty – posloupnosti hran
- deadlock nastane, když proces detekuje lokální cyklus

**edge-pushing**

- cirkulují zkušební zprávy
- blokovaný proces posílá zkušební zprávu procesům držícím požadované zdroje
- deadlock nastává pokud iniciátor obdrží vlastní zprávu

## Algoritmus Rosenkranze, Stearnse a Lewise

- pracuje na základě porovnání časových razítek, přidělených v době vzniku procesů
- **wait-die -** Je-li proces P2 starší než P1, pak P2 čeká dokud P1 prostředek neuvolní. Je-li P2 mladší než P1, je P2 zrušen a musí se provézt znovu se stejným časovým razítkem.
- **wound-wait -** Pokud je P2 starší než P1, pak je P1 zrušen. Je-li P2 mladší než P1, pak P2 čeká na uvolnění prostředku procesem P1. Hlavní zásadou, podle které se přidělení žádaného prostředku řídí je, že starší proces nikdy na prostředky držené mladším procesem nečeká.

# 9. Distribuované sdílené paměti, replikace, modely konzistentnosti

- žádný procesor nemůže přímo přistupovat do paměti jiného (cizího) procesoru - **NORMA (No Remote Memory Access)**
- **NUMA - No Uniform Memory Access** - procesor přímo adresuje položky v paměťovém prostoru, přičemž jednotlivé stránky mohou být libovolně distribuovány mezi paměťmi různých procesorů
- **Distribuovaná sdílená paměť** (DSM) - abstrakce používaná pro sdílení dat mezi procesy v počítačích, kde není sdílená fyzická paměť.
- Pouze model typu peer-to-peer
- z důvodu rychlého přístupu je v každém uzlu lokální kopie dat, systém musí zajistit konzistentnost dat v jednotlivých uzlech.
- **Těsně vázané multiprocesory se sdílenou pamětí** – propojení pomocí sběrnice, max. 10 až 20 procesorů, architektura NUMA (Non-Uniform Memory Access) s max. 64 procesory, procesory vidí jeden adresní prostor obsahující všechny paměti na všech deskách.
- **Volně vázané multiprocesory s distribuovanou (sdílenou) pamětí** – nemají společný paměťový prostor, ale jsou propojeny velmi rychlou sítí s přepínači.

### *Přístupy k implementaci distribuované sdílené paměti*

- Distribuovaná sdílená paměť může být realizována specializovaným hardware, konvenční stránkovou sdílenou pamětí, middleware, nebo jejich kombinací.

## Modely konzistentnosti

- **Konzistentnost** - soudržnost paměti a programů při manipulaci s daty.

### *Striktní konzistence (strict consistency)*

- Jakékoliv čtení z paměti z adresy *x* vrátí hodnotu uloženou při posledním zápisu na adresu *x*. Striktní konzistence je nejsilnější, je přirozeně zajištěna na jednoprocesorových systémech.
- v distribuovaném systému však nedosažitelné.

### *Sekvenční konzistence (sequential consistency)*

- Výsledek jakéhokoliv výpočtu je stejný jako kdyby všechny operace všech procesorů byly vykonávány v nějakém sekvenčním uspořádání a operace každého jednotlivého procesoru jsou vykonávány v pořadí specifikovaném programem.

### *Kauzální konzistence (causal consistency)*

- Zápisy, které jsou potenciálně kauzálně vázané, musí být viděny všemi procesy ve stejném pořadí.
- Konkurenční zápisy mohou být viděny v různém pořadí.

### *PRAM konzistence (PRAM consistency)*

- Zápisy prováděné jedním procesem jsou viděny ostatními procesy v tom pořadí, ve kterém byly prováděny, avšak zápisy různých procesů mohou být viděny různými procesy různě.

### *Slabá konzistence (weak consistency)*

- nechat proces ukončit kritickou sekci a poté zajistit rozeslání změn všem ostatním procesům.

### *Uvolňovací konzistence (release consistency)*

- dvě operace - acquire (požadavek) a release (uvolnění).
  - Před běžným přístupem ke sdílené proměnné musí být úspěšně ukončeny předchozí požadavky procesu.
  - Před provedením uvolnění musí být ukončeny všechny předchozí zápisy i čtení prováděné procesem.
  - Požadavky a uvolnění musí být PRAM konzistentní.
- **Eager release consistency** - po *release* se propagují změny všem procesům.

- **Lazy release consistency** - *po release* se nic nepropaguje, až při *acquire* jiného procesu - časové značky.

### *Přístupová konzistence (entry consistency)*

- Požadovaný přístup procesu k synchronizační proměnné není povolen dokud nebyly provedeny všechny aktualizace chráněných sdílených dat procesu.
- Exklusivní přístup procesu k synchronizační proměnné je povolen pouze v případě, že žádný jiný proces nepřistupuje k synchronizační proměnné, a to ani neexklusivně.
- Po exkluzivním přístupu k synchronizační proměnné si příští neexkluzivní přístup libovolného procesu k synchronizační proměnné musí vyžádat aktuální kopii dat od vlastníka synchronizační proměnné.

### *Shrnutí konzistenčních modelů*

| | Konzistence | Vlastnosti |
|---|---|---|
| a | Striktní | Absolutní časové uspořádání |
| | Sekvenční | Všechny události jsou vidět ve stejném pořadí |
| | Kauzální | Kauzálně vázané události jsou vidět ve stejném pořadí |
| | PRAM | Události jednoho procesu vidět ve stejném pořadí |
| b | Slabá | Sdílená data jsou konzistentní po synchronizaci |
| | Uvolňovací | Sdílená data jsou konzistentní po opuštění kritické sekce |
| | Přístupová | Sdílená data vázaná na kritickou sekci jsou konzistentní při vstupu do kritické sekce |

## Replikace

- Zvýšení spolehlivosti.
- Zvýšení výkonnosti.
- Nutnost zachování škálovatelnosti systému co do počtu komponent i geografické rozlehlosti.
- Výkonnost se snižuje, protože s opravou jedné kopie souvisí i oprava ostatních kopií pro zachování konzistentnosti.
- **Potenciální problémy:**
  - Šířka pásma požadovaná pro aktualizaci kopií může být velká.
  - Udržování konzistentních replikovaných kopií může být problémové z pohledu škálovatelnosti.
- Dodržování dohod při replikaci dat:
  - Vícenásobné kopie.
    - zvyšují výkonnost a redukují dobu přístupu
    - zvyšují také režii pro udržení konzistentnosti
  - Zdrojem řešení je udržování konzistentnosti
    - volba vhodné sémantiky
    - těsná konzistentnost vyžaduje globálně synchronizované hodiny
  - Řešení: snížíme požadavky na konzistentnost
    - jsou k dispozici různé stupně konzistentnosti
- **Distribuované filesystemy** často umožňují replikaci souborů, tj. udržování více kopií všech nebo jen některých souborů na více fileserverech. To má několik dobrých důvodů:
  1. Spolehlivost (reliability) - při výpadku jednoho serveru nejsou ztracena data
  2. Dostupnost (availability) - k souborům lze přistupovat i v případě výpadku jednoho nebo několika serverů
  3. Výkon (performance) - lze přistupovat k nejbližším datům, výkon fileserveru je rozdělen mezi několik serverů
- **explicitní replikace** - uživatel se sám stará o udržování konzistence
- **odložená replikace** - zápis do primární repliky, aktualizace sekundárních replik později

- **skupinová komunikace** - zápisy jsou simultánně zasílány všem dostupným replikám

### *Aktualizační protokoly*

- zajištění konzistence replik
- **primární replika** - všechny změny se provádějí na jedné vybrané replice a tento server se sám stará o aktualizaci ostatních.
  - nevýhoda - v případě nedostupnosti primární repliky nelze s objektem pracovat.
- **hlasování (voting)** - získání dostatečného počtu hlasů pro povolení přístupu k objektu
  - **většinové hlasování** (majority voting). - pro jakoukoliv operaci je třeba získat většinu (nadpoloviční množství) z celkového počtu serverů.
    - Při zápisu se všem zapsaným replikám přiřadí nové číslo verze (všem replikám stejné).

# 10. Distribuované souborové systémy a jejich vlastnosti

- **Souborové služby** – dostupné funkce. Akce, které jejich volání způsobí, neříkají nic o jejich realizaci.
- **Souborový server** – proces, který realizuje souborové služby.
- **Distribuovaný systém souborů** - odděluje souborové služby a adresářové služby

## Manipulace se soubory

- Zobrazení souboru:
  - posloupnosti slabik (UNIX, MS-DOS)
  - sekvence rekordů (databázové systémy)
  - B-strom, tabulka s přímým přístupem.
- atributy souboru (vlastník souboru, velikost souboru, datum vytvoření, přístupová práva a pod.)
  - nejsou součástí souboru
- ochrana souborů:
  - ACL (uživatel - právo)
- přístupu k souboru:
  - s přesunem souboru (upload / download) – přesuny celých souborů
  - se vzdáleným přístupem (remote access) – přesuny pouze částí, nutná stálá komunikace

## Rozhraní pro manipulaci s adresáři

- Manipulace s adresáři zahrnuje následující operace
  - vytváření a rušení adresářů
  - nastavení jména a přejmenování souborů
  - přesuny souborů mezi adresáři
- hierarchický systém souborů

možnost vytvářet linky - jeden výskyt souboru/adresáře má několik jmen

## Požadavky na DFS - Transparentnost jmen

- hiearchické adresáře - jména nejsou zcela transparentní vzhledem ke svému umístění na jednotlivých uzlech sítě

## Dvouúrovňové označování

- Symbolické jméno - označení souboru ”pro uživatele”
- binární jméno - identifikátor souboru
- Úkolem adresářových služeb je mapování symbolického jména na identifikátor souboru.

## NFS (Network File System)

- Fa SUN
- transparentní vzdálený přístup ke sdíleným souborovým systémům prostřednictví počítačové sítě
- **NFS protokol** - nezávislý na typu počítače, operačním systému, síťové architektuře a transportním protokolu -> zajištěno použitím mechanizmu volání vzdálených procedur
- Doplňkový **protokol MOUNT** - funkce specifické pro příslušný operační systém, které umožňují klientu připojit vzdálený strom adresářů do určitého místa v lokálním souborovém systému
- modul pro uzamykání souborů NLM (Network Lock Manager) - izoluje stavové aspekty zamykání souborů v odděleném protokolu.
- **NIS** (Network Information Service) - doplněk, který zajišťuje jednotnou administraci jmen a hesel uživatelů
- NFS lze použít i pro bezdiskové počítače, kdy při inicializaci operačního systému se inicializuje i jeho souborový systém, uložený na vzdáleném serveru.
- Soubor musí být uložen na jednom počítači, nelze jej rozdělit na více částí a rozmístit po síti.

- Klient musí podporovat VFS (Virtual File System), dovolující mapování vzdálených souborů do lokálního souborového systému
- NFS server pracuje jako bezestavový

## AFS (Andrew File System)

- Cíle:
  - maximální unifikace pracovního prostředí uživatele, nezávislost na umístění uživatele a souborů
  - transparentnost pro uživatele v prostředí UNIXu
  - spolehlivost, odolnost proti poruchám, transparentnost údržby, zálohování a rozšiřování
  - velký výkon (snížení zátěže sítě a serverů), nasazení v univerzitním prostředí velkého rozsahu s možností libovolného růstu
  - zachování běžných výhod sdíleného souborového systému (pro uživatele, programové systémy i administrátory)
- Používá síťové protokoly TCP/IP.
- Při návrhu se vychází z následujících myšlenek:
  - kompatibilita s UNIXem na úrovni jádra
  - základní jednotkou pro manipulaci s daty je soubor
  - rozdělení systému na více menších serverů

### *Manipulace se soubory*

- celý soubor přenese na stanici, kde se s ním pracuje
- Na server se přenáší pouze po uzavření souboru, po volání close nebo sync
- Pokud dojde k výpadku serveru, přejde uzel do lokálního režimu a všechny soubory, které byly umístěny do vyrovnávací paměti jsou nadále k dispozici.

### *Struktura serverů*

- Systém je rozdělen do relativně malých serverů -> lepší poměr výkon/cena a vyšší spolehlivost
- Všechny řídicí informace jsou uloženy v distribuovaných databázích.

### *Přístupová práva*

- Seznamy přístupových práv (Access Control List).
- obsahuje dvojice subjekt (uživatel, skupina), přístupová práva.
- Seznam přístupových práv se dědí.

### *Ověřování*

- ověřování pomocí serverů KERBEROS.
- Tiket používán pro komunikaci s dalšími servery AFS.

### *Souborový systém z pohledu AFS klienta*

- strom adresářů, kořenem je /afs, kam se souborový systém připojí při inicializaci klienta

### *Souborový systém z pohledu AFS serveru*

- souborový systém rozdělen do svazků - skupina souborů a adresářů logicky k sobě patřících
- svazek má přiřazen limit své velikosti (quota).
- Každý svazek, určený pouze pro čtení, může mít několik replik, rozmístěných libovolně na různých serverech -> Při mapování se automaticky volí nejsnadněji přístupná replika z funkčních

### *AFS versus NFS*

- AFS má jmenný prostor souborů, NFS je závislé na připojení.
- Soubory AFS mohou být libovolně rozmístěny po serverech
- AFS vzhledem k použitému mechanizmu vyrovnávací paměti zvládne velké rozšiřování.
- AFS má zabudovaný mechanizmus bezpečnosti
- AFS umožňuje replikaci dat označených jako data pro čtení -> spolehlivost, dostupnost dat.

# 11.Chyby v distribuovaných systémech, spolehlivost

## Odolnost proti poruchám

- **partial failure** – částečná chyba
- **error isolation** – ostatní komponenty nejsou zasaženy
- automatická obnova z částečných chyb

## Synchronní a asynchronní systémy

- **synchronní systém** - systém reaguje do stanoveného časového intervalu
- **asynchronní systém** - není zaručen časový interval, během kterého musí systém odpovědět

**Odolnost proti poruchám (spolehlivost) závisí na**

- **dostupnost** (availability)
  - část doby, během které systém splňuje svou specifikaci
- **spolehlivost** (reliability)
  - Pravděpodobnost že systém selhal během dané doby.
  - Typicky používané pro popis systémů, které nelze opravit nebo kde je kritický spolehlivý nepřetržitý provoz systému
- **bezpečnost** (safety)
  - Jestliže systém dočasně selže, jeho specifikace se přizpůsobí a nic katastrofického se nestane
- **udržovatelnost** (maintainability)
  - Míra jak snadno je možné systém opravit

## Základní pojmy

- **error** (chyba, omyl, odchylka) – část stavu systému, která může vézt k poruše.
- **fault** (porucha, nedostatek, chyba) – chyba ve vnitřních stavech komponent systému nebo v návrhu systému
- **failure** (selhání) – odchylka z chování, které je popsané v jeho specifikaci
- **erroneous state** (chybový stav) – takový vnitřní stav systému, při kterém existují okolnosti, za nichž další zpracování normálním algoritmem povede k selhání (failure), které není přisuzované následující poruše.

Závislosti: Fault (porucha) → error (chyba) → failure (selhání)

## Typy poruch

**Stálé poruchy** (Hard faults)

- stálá (permanent) – pokračování po opravě komponenty – programové, technické chyby
- výsledná selhání jsou označována jako "tvrdá"

**Dočasné poruchy** (Soft faults)

- přechodné nebo občasné
- přechodná (transient) – objeví se a zmizí – opakování operací
- občasný (intermittent) – obtížně diagnostikovatelný – chybný kontakt
- představují více než 90% všech chyb
- výsledná selhání jsou označována jako "měkká"

## Zpracování chyb

- **prevence** (preventing)
- **odstranění** (removing)
- **předpovídání** (forecasting)

**Odolnost proti poruchám (fault tolerance) -** systém může provádět služby, vedoucí k prevenci chyb

## Typy selhání (failure)

- **zhroucení** (crash) – server se zastaví, dokud se nezastavil, pracoval dobře
- **vynechání** (omission) – server neodpoví na příchozí požadavek
  - chybuje při příjmu
  - chybuje při posílání
- **časování** – reakce je mimo časový interval
- **chybná reakce** – server odpovídá nekorektně
  - chybná hodnota
  - chybná změna stavu (odchylka od korektního postupu řízení)
- **svévolná** (arbitrary) – libovolné odpovědi v libovolnou dobu – Byzantinské chyby

**Fail silent systems** - server nedává najevo své problémy, ostatní mohou reagovat nekorektně (zastavení serveru)

**Fail-safe server (zabezpečený proti poruše)** - server produkuje náhodné výstupy, rozpoznatelný ostatními procesy

## Maskování chyb a redundance

- cílem je skrýt výskyt selhání (failure)
- **informační (datová) redundance** – replikace nebo redundantní kódování dat - obnova (Hammingovy kódy), parita v paměti,
- **časová redundance** – násobné provádění
  - opakování přenosu. timeout
- **fyzikální redundance** – komponenty navíc – hardware, software
  - práce s redundantními komponentami, nikoliv s daty
  - odolnost vůči ztrátě nebo poruchám komponent
  - násobné fyzické komponenty (Pn = pn)

## Process resilience (pružnost, elastičnost)

- ochrana proti selhání procesů – replikace procesů do skupin
- organizování několika identických procesů do skupin
- zprávy ve skupině přijímají všechny procesy
- skupina může být
  - statická
  - dynamická (join, leave, destroy – group management
- proces může být členem několika skupin
- organizace skupiny může být
  - plochá (flat) – neexistuje centrální prvek, komplikované
  - hierarchická – ztráta koordinátora znamená zhroucení
- členství ve skupinách
  - server skupiny – jednoduché, centrální prvek
  - distribuované – spolehlivé skupinové doručování
- problém – opuštění skupiny – není indikováno zhroucení
  - testování přítomnosti procesů, ale zpožděná odpověď
  - opouštění (leave) a připojování (join) musí být synchronní
  - znovuvytvoření skupiny – může iniciovat jakýkoliv proces

## Maskování poruch a replikace

- skupina procesů – může maskovat poruchy (identické procesy) – replikace procesů
- dva způsoby řešení
  - primary backup protocols
  - replicated write protocols

**Primary backup protocols**

- hierarchické uspořádání (primary, backup)

- primary koordinuje všechny operace zápisu
- výpadek se indikuje
  - primary - periodickým vysíláním zprávy (žiji)
  - backup – dotazovací zprávou (žiješ?)
  - problém s nastavením timeoutu (asynchronní systém)
- pokud se primary zhroutí, backup – vyvolání algoritmu výběru – volba nového primary
- procesy jsou organizovány hierarchicky

**Replicated write protocol**

- používá aktivní replikaci nebo protokoly založené na hlasování
- ploché (flat) skupiny – protokoly založené na hlasování pro realizaci oprav
- neexistuje úzké místo vzhledem k chybám

## Skupiny procesů a tolerance chyb

- systémy s replikací zápisu – pouze zapisují (zjednodušené)
  - systém je k-odolný proti poruchám – může přežít chybu v k-komponentách
  - stačí k+1 komponent – k nefunguje, jeden stačí
- Byzantinské chyby
  - pro systém k-odolný proti poruchám musí být 2k+1 procesů
  - k – produkuje chybnou odpověď
  - k+1 – produkuje správnou odpověď
  - řešení pomocí hlasování a majority
  - neví se ale kolik procesů selhalo

## Dohoda v systémech s poruchami

- náhrada hlasování
- replikované procesy mají dosáhnout dohody
  - výběr koordinátora
  - rozhodnutí o provedení/zrušení transakce
  - rozdělování úloh mezi dělníky
  - synchronizace
- základní cíl
  - bezchybné procesy musí dosáhnout dohody v konečném počtu kroků
- slavný problém – problém dvou armád
  - dosažení dohody pomocí jednobitové informace
  - dosažení dohody o započetí útoku
  - komunikace nespolehlivým kanálem
- dva základní případy pro řešení
  - generálové – bez chyby, komunikace nespolehlivá – nemá řešení
  - generálové s chybami, komunikace spolehlivá – problém Byzantinských generálů
  - výsledek – pro 3m+1 procesů musí být 2m+1 korektních
  - není-li spolehlivá komunikace, musí být všechny procesy v pořádku

### *Byzantinští generálové*

Problém Byzantinských generálů pro tři věrné a jednoho zrádce.

- Generálové oznamují ostatním sílu svých baterií (v jednotkách 1000 vojáků).
- Každý generál vytvoří vektory vytvořené generály na základě přijatých údajů a pošle ho ostatním.
- Každý generál má vektory od všech ostatních, hlasováním zjistí správné hodnoty.

## Obnova po chybě

- redundantnost pomáhá odstranit chyby "za běhu"
- chyby se stávají transparentní vzhledem k okolí
- netransparentní zajištění systému proti chybám – obnova po chybě (failure recovery)

- kontrolní body (checkpoint) – periodické ukládání stavu
  - koordinované (synchronizované hodiny, blokování)
  - nekoordinované
- logování (logging) – zaznamenávání operací se stavem
  - synchronní, asynchronní

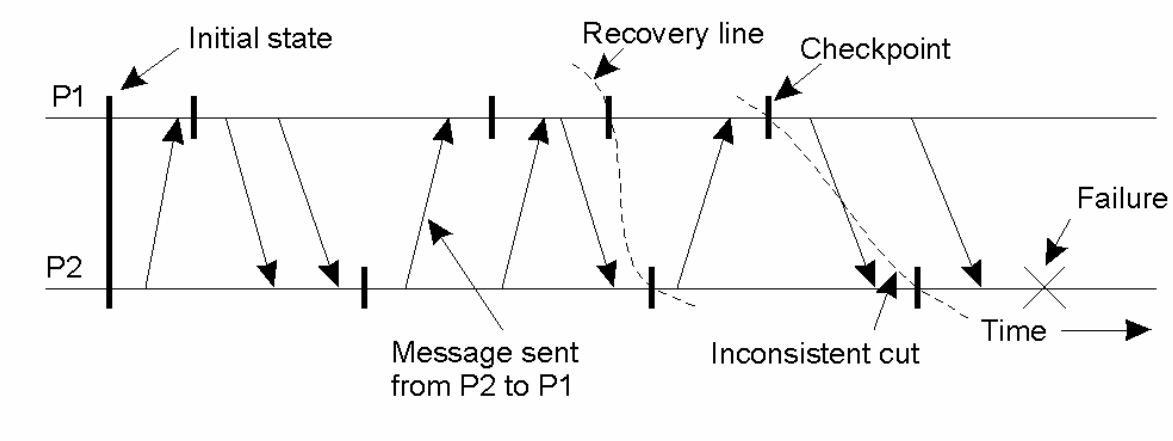
## Proces obnovy

- host obnovený do předchozího stavu musí opakovat všechny operace až do místa chyby
  - musí poslat duplicitní zprávy
  - ostatní musí být schopni rozpoznat duplicitní zprávy a zahazovat je
- ostatní host systémy se také musí vrátit do předchozího stavu (kaskádní rollback)
- všechny systémy se musí dostat do téhož stavu (recovery line)

## Incarnation Numbers (etapa)

- každé období je charakterizováno sekvenčními inkarnačními čísly
- jsou přítomna v každé zprávě
- v systému jsou zapamatována ve stálé paměti
- pokud se systém probouzí (převtěluje), zasílá všem nové inkarnační číslo
  - < - duplicitní zpráva, zahodit
  - > - čekej na zprávy pro obnovu
  - = zpracuj zprávu

## Kontrolní body



# 12. Migrace kódu a procesů

- Hlavní důvody – výkonnost a flexibilita
- Migrace procesu (silná mobilita)
    - Zlepšení výkonu celého systému – lepší využití celosystémových zdrojů
- 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

## Flexibilita

- Dynamická konfigurace distribuovaného systému
- Klienti nepotřebují přeinstalovaný software – zavedení programu na žádost

## Modely migrace

- Proces je dán
    - Segmentem kódu
    - Segmentem zdroje dat
    - Segmentem provedení – procesor
- slabá / silná mobilita
    - Slabá – přenesený program začíná počátečním stavem
- Iniciovaná vysílajícím / příjemcem
    - Iniciovaná vysílačem (kód se přenáší z odesílatele)
        - Klient posílá požadavek na DB server
        - Klient by mohl být registrován předem
    - Iniciovaná příjemcem
        - Java Applety
        - Příjemce může být anonymní

## Migrace entity

- Migrace kódu
    - Provedeno odděleným procesem, Applety – provede se v cílovém procesu
- Migrace procesu
    - Vzdálené klonování

## Modely migrace kódu

**Mechanizmus mobility**

- Slabá mobilita
    - Inicializace Odesílatelem
        - Provedení v cílovém procesu
        - Provedení odděleným procesem
    - Inicializace Příjemcem
        - Provedení v cílovém procesu
        - Provedení v odděleném procesu
- Silná mobilita
    - Inicializace Odesílatelem
        - Migrace procesu
        - Klonování procesu
    - Inicializace Příjemcem
        - Migrace procesu
        - Klonování procesu

## Migrace zdrojů

**Závisí na propojení procesu a zdroje (bind)**

- **Podle identifikátoru** - Webová stránka, FTP server
- **Podle hodnoty** - Java knihovny
- **Podle typu** - Tiskárny, lokální zařízení

**Závisí na typu spojení (attach)**

- Nepřipojený k uzlu - Datové soubory
- Svázané zdroje (může být přesunut, ale za vysokou cenu) - Databáze, webové stránky
- Fixní zdroje - Lokální zařízení, koncové body komunikace

## Migrace virtuální paměti

- **Zmražení a kopírování** – pozastaví proces, kopíruje všechny paměťové stránky, vyřeší propojení, startuje proces v novém hostu
  - *Výhody*: čisté, žádná další závislost
  - *Nevýhody*: kopírování zbytečných stránek, zmražení během kopírování
- **Metoda předběžného kopírování** – proces pokračuje v činnosti pokud nejsou stánky překopírovány, pak se zmrazí a kopírují se modifikované stránky
  - *Výhody*: žádná zbytková závislost
  - *Nevýhody*: některé stránky se kopírují dvakrát
- **Líná migrace** – proces migruje bez přesunu stránek. Stránky migrují když je to potřeba
  - Výhody: rozprostření nákladů na migraci, nepotřebné stránky nejsou kopírovány
  - Nevýhody: vytváří zbytkové závislosti
- **Migrace s distribuovaným systémem souborů** - paměť procesu existuje jako paměťově mapovaný soubor v DFS. Při migraci jsou špinavé bloky zapsány a soubor je mapován z nového hosta
  - Výhody: migrace je rychlá a bez zbytkové závislosti
  - Nevýhody: stránkování je drahé – komunikační čas, čas přístupu

## Migrace komunikačních kanálů

- Pokud proces komunikuje přes interprocess komunikaci, musí také komunikace migrovat
- Jednou z možností je informovat všechny strany spojené s komunikací o novém umístění
  - *Výhody*: žádné zbytkové závislosti
  - *Nevýhody*: obtížné sledovat všechny zúčastněné procesy, musí se posílat dost zpráv
- Jiné řešení pomocí **redirekce**
  - *Výhody*: jednoduše implementovatelné, levné
  - *Nevýhody*: zbytková závislost
- Hybridní řešení, používá nejdříve redirekci a informuje účastníky o novém umístění pro další komunikaci

## Migrovat nebo nemigrovat

Migrující proces představuje velké náklady, které závisí na:

- Jestliže je host náhle přetěžován, může migrace pomoct
- Procesy s významnou meziprocesovou komunikací nebo virtuální pamětí nejsou vhodné pro migraci
- Pro migraci jsou vhodné procesy s dlouhou dobou běhu, které rozprostřou náklady spojené s migrací na delší dobu
- Na počítači nezávislý kód je méně nákladný pro migraci

# 13. Bezpečnost v distribuovaných systémech, ověřování uživatelů, ověřovací servery

### *Počítačová síť – otevřený systém*

- napadení sítě
- napadení počítačů
- odposlech přenášených informací

### *Způsoby napadení sítě*

- **pasivní**
  - odposlech (data, hesla)
  - analýza přenosu (odkud kam, délka, atd...)
- **aktivní**
  - modifikace – změna obsahu, adresy, pořadí atd...
  - zadržování – opožděné posílání zpráv
  - zahlcení požadavky
- **cíle obrany**
  - prevence pasivního útoku
  - detekce aktivního útoku

### *Typy ochran*

- **ochrana zdrojů** - ochrana proti neautorizovanému použití prostředků v operačním systému
  - přístupovou maticí
  - přístupovým seznamem
  - seznamem schopností
- **bezpečná komunikace** - vlastní ochrana přenášené informace
  - zaměření na linku – šifrování linky
  - zaměření na koncové uživatele – šifrování až na koncových bodech
  - zabezpečení na úrovni spojení
- **ověřování uživatelů** – zabezpečení, aby zprávy přicházely od ověřeného zdroje a byly přenášeny bez modifikace
  - Kerberos

## Způsoby neautorizovaného přístupu

Neautorizované přístupy k datům můžeme rozdělit do čtyř kategorií.

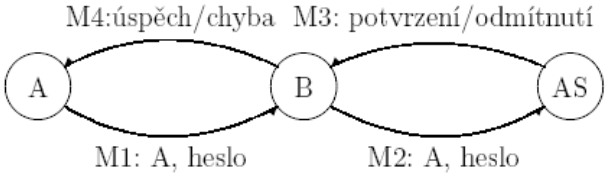
1. **Únik informace** (Leaking) - intruder má komplice, legálního uživatele, který mu předává informace.
2. **Listování** (Browsing) - intruder se pokouší o čtení informace, uložené v systému. Data nemodifikuje.
3. **Odvozování** (Interferenting) - intruder čte informace a snaží se o odvození informace aplikací odvozeného klíče (např. porovnáním šifrované a otevřené informace).
4. **Maskování** (Masquerading) - intruder se tváří jako autorizovaný uživatel nebo program. Typickým představitelem tohoto způsobu napadení je Trojský kůň.

## Šifrování

- **Symetrické šifrování**
  - kódování i dekódování využitím pouze 1 klíče (DES, AES, IDEA).
- Asymetrické šifrování (př. SSL)
  - využití 2 klíčů - 1 klíč je veřejný a druhý je tajný
  - veřejným klíčem je zpráva kódována a privátním klíčem se provádí dekódování

## Ověřování pravosti uživatele

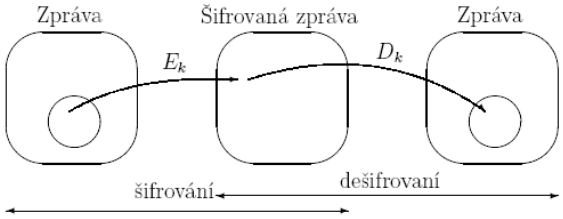
- **Ověřovací servery**
  - slouží k ověření pravosti uživatele
  - lepší utajení klíčů, hesla se přenáší v šifrované podobě
  - KDC (Key Distribution Center) – databáze klíčů (indexovaná podle jmen uživatelů)
- **Ověřování -** schémata:
  - alespoň jedno tajemství a schopnost rozpoznat jeho správné použití
  - ověřovací metody:
    - **jednoduché** (hesla)



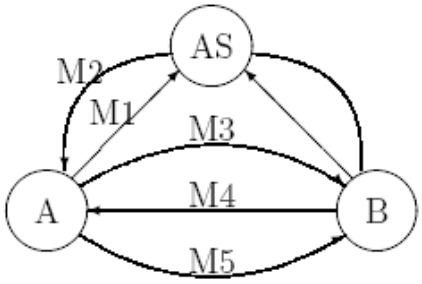

    - **přísné** (založeny na šifrovacích metodách)

### *Přísné metody*

- použití symetrických




a nesymetrických kódů



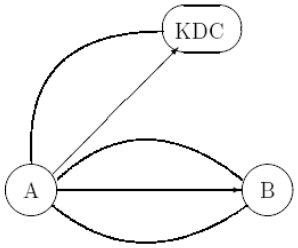

- přenos dat šifrovaně
- založené na ověřovacích serverech
- založené na protokolech s minimální znalostí
  - uživatel dokazuje svoji identitu odpovídáním na šifr. otázky serveru
    klient posílá serveru zprávu M1: {R, ID}
    klient posílá serveru zprávu M2: {C}K
    klient posílá serveru zprávu M3:{f(C) }$_K$
    R … požadavek, K … tajný klíč, C … náh. číslo, f(C) … domluvená funkce

## Metody rozdělování klíčů

Rozdělování klíčů v distribuovaných systémech dovoluje entitám průběžně přidělovat klíče (průběžně měnit klíče a tím znížit možnost detekce klíče).

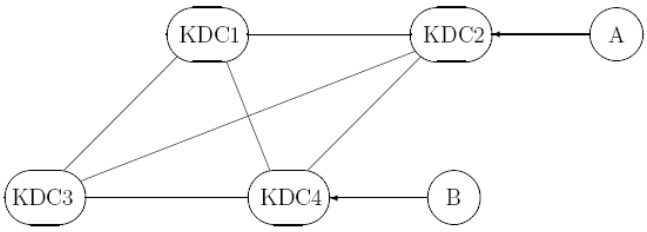
### *Centralizovaná distribuce klíčů*

- Centralizovaná distribuce klíčů je nejednodušší, představuje však úzké místo v systému.




### *Plně decentralizovaná distribuce klíčů*

- Každý hostitelský systém slouží jako KDC.
- Vzájemně se informují o přidělování klíčů.
- Výhody: každý požadavek na přidělení klíče je lokální, komunikace mezi hostitelskými systémy je šifrovaná a spolehlivá.
- Nevýhody: příliš mnoho komunikací, potřebných pro distribuci klíčů.




### *Hierarchická distribuce klíčů*

tří skupiny:

- globální KDC mezi různými oblastmi
- regionální KDC v nějaké oblasti
- lokální KDC pro lokální přidělení.

## Digitální podpisy

- vlastnosti:
  - unikátní
  - nepadělatelný
  - ověřitelný
  - nezapřitelný
  - jednoduché metody
- realizace dvojím způsobem:
  - přímé ověřování podpisu
  - nepřímé ověřování podpisu (existuje třetí entita, která řeší spory)
- založeny jak na symetrických, tak i nesymetrických klíčích.
- lze použít pro ověřování zpráv a ověřování uživatelů.