Přednášky z distribuovaných systémů

Jiří Ledvina

2002

Obsah

Kapitola 1

Úvod

K budování distribuovaných výpočetních systémů existují následující důvody:

1. zvýšení spolehlivosti vytvořením vícenásobných zdrojů
2. zvýšení průchodnosti rozdělením zdrojů do více uzlů
3. zvýšení dostupnosti

Distribuované systémy se budují podle dvou základních modelů.

Serverový model předpokládá jednoznačné rozdělení funkcí na toho, kdo je poskytuje (server) a toho, kdo je vyžaduje (klient). Toto nesymetrické uspořádání má řadu výhod i nevýhod. Výhoda je v optimalizaci technického i programového vybavení pro daný účel. Nevýhoda v nezastupitelnosti role v případě jeho výpadku.

Druhým modelem pro distribuované systémy je integrovaný model, který předpokládá, že není třeba explicitně rozlišovat servera a klienta, že je tato role dána okamžitými potřebami jednotlivých procesů.

Definice: Distribuovaný výpočetní systém je takový systém, ve kterém jsou komponenty systému navzájem propojeny sítí tak, že uživatelům nabízí jedno sourodé počítačové prostředí.

Proč se rozkládá výpočetní výkon v síti na samostatné prvky

1. co lze rozložit
2. co to způsobí
    - výkon
    - spolehlivost
    - dostupnost

Někdy to jinak řešit nelze - např. databáze jmen v Internetu. Distribuce umožní snížení ceny, zlepšení využití zařízení, využití speciálních zařízení (superpočítače).

Distribuce

- výpočetní výkon - distribuované operační systémy

- data - distribuované databázové systémy
  - distribuované informační systémy
  - rozdělení dat v síti, replikace dat (vícenásobné kopie)

Kapitola 2

Modely distribuovaných systémů

V zásadě lze rozeznat dva základní modely distribuovaných systémů.

- model klient - server
- integrovaný model

V modelu klient/server nejsou standardní služby OS implementovány na každém uzlu, ale na specifických uzlech - serverech. Klienti obsahují pouze programové vybavení pro přístup k serveru. Nevýhodou je, že služby poskytované za hranicí klienta jsou zatíženy režií, spojenou s vyvoláním vzdálené služby. Klient pracuje pouze jako programový interface pro server.

V integrovaném modelu je každý uzel implementován s kompletní verzí OS, tzn. že v každém uzlu je přítomen kód pro realizaci požadovaných funkcí. Většina funkcí je realizována lokálně, hranice systému se překročí pouze tehdy, pokud je to nutné. Pojem kompletní verze je nutno chápat jako minimální domluvenou množinu funkcí, podporovanou všemi uzly.

2.1 Síťové servery

Počítačová síť může být použita k realizaci mechanismů, které umožní sdílení zvláště drahých zařízení mezi uživateli sítě nebo poskytování "drahých" služeb. Základem mnoha strategií pro sdílení zdrojů je vyčlenění speciálního uzlu se známou (všeobecně známou) adresou, který provádí služby pro celou síť. Tento uzel se označuje jako server. Uzly, které služby využívají se nazývají klienti. V síti může existovat několik serverů, které provádějí shodné služby. Uzel, chovající se jako server k jednomu uzlu se může chovat jako klient k jinému uzlu.

Servery se dají realizovat různými způsoby. Podle toho se liší svojí složitostí a použitelností.
Podle ochrany uživatele se dělí na

- monouživatelské
- víceuživatelské

Podle způsobu zpracování požadavku se dělí na

- interaktivní (jednorázové zpracování požadavku serverem, dovoluje zpracovávat pouze jeden požadavek v čase)

- procesové (vytvoří se proces pro zpracování požadavku, umožňuje paralelní zpracování více požadavků)

Podle způsobu uchování stavu rozpracovanosti požadavku se dělí na

- stavové (pamatuje si stav rozpracování pžadavku)
- bezestavové (stav je předán klientu spolu s odpovědí, zpět přenášen spolu s dalším požadavkem klienta)

Podle používaných komunikačních služeb se dělí na servery využívající

- nespojované služby (spolehlivé nebo nespolehlivé datagramové služby)
- spojované služby (virtuální okruhy)

2.2 Typy serverů

Abychom přiblížili problematiku realizace distribuovaných systémů, uvedeme v této kapitole funkční popis některých známých typů serverů.

- obecný popis který příliš nepřihlíží k obecným problémům distribuce
- obecné principy budou rozebrány později
- výklad podle klasického pojetí, bez důrazu na distribuovanost

2.2.1 Časový server

Časový server slouží k přenosu informace o přesném čase. Předpokládá se propojení klienta se serverem pomocí počítačové sítě. Problém spočívá zejména v tom že při přenosu informace mezi dvěma počítači dochází ke zpoždění, které není konstantní. Závisí zejména na zatížení sítě a je proto těžko odhadnutelné. Podle požadavků na přesnost nastavení času bylo vyvinuto několik protokolů.

V nejjednodušším případě jde pouze o přenos aktuálního času časového serveru. Jedná se v principu o dotaz typu "kolik je asi hodin". Takovýto časový server je velmi jednoduchý, realizuje se jako bezestavový a interaktivní.

Složitější systémy vyžadují nejen znalost přesného času, ale i časovou synchronizaci s dalšími počítači v síti s přesností na ms (a více). Vyžadují, aby v každém ze spolupracujících počítačů byl jednotný čas. To dovoluje např. realizovat decentralizované překlady a sestavení programů pomocí programu make (nová verze se odvozuje od času vzniku), ověřování uživatelů (Kerberos), přesná měření ve vzdálených místech a pod. K tomu se používají složitější protokoly, vycházející z definované stability lokálního časového zdroje a odhadu jeho maximální možné odchylky. 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. Klienti časových serverů provádí vlastní kvalifikovaný odhad

přesného času. Při zjištění odchylky jsou lokální hodiny zrychleny nebo zpomaleny tak, aby nedošlo k časové diskontinuitě, ale k postupnému dosažení přesného času (např. čas nelze vrátit zpět).

2.2.2 Jmenný server

Jmenný server je dnes již klasickým příkladem distribuované databáze, používané v Internetu pro převod jmen na adresy počítačů a opačně. Síť je rozložena na jednotlivé administrativní domény, ve kterých je registrace jmen prováděna centralizovaně. Tyto domény vytváří hierarchckou strukturu. Jmenné servery, umístěné v jednotlivých doménách a provádějící převody, mají k dispozici převodní tabulky lokální administrativní domény a odkazy na jmenné servery nadřazené domény. Pokud není převod uveden v lokálních tabulkách, vyšlou dotaz jmennému serveru nadřazené domény. Tento server buď provede převod sám, nebo sdělí jméno servera, který by mohl převod provézt.

Důvod decentralizace tabulek do jednotlivých domén je zřejmý. Udržovat tyto tabulky centralizovaně nelze z důvodu přetížení a udržení konzistentnosti. Centralizovaná databáze by byla vystavena velkému počtu dotazů i velkému počtu požadavků na provedení změn (přidávání a rušení počítačů). To by vedlo k přetížení vlastního serveru i komunikačních cest k němu.

2.2.3 Ověřovací server

Ověřovací server slouží k ověřování totožnosti uživatele a k určování jeho práv přístupu ke zdrojům sítě.

2.2.4 Tiskový server (printer server)

Tiskový server dává možnost připojit nákladné tiskárny k personálním počítačům, propojeným po počítačové sítě. Tiskový server může být realizován jako

- vyčleněný tiskový server (stanice)
- proces (příp. rezidentní program) - aplikační program (např. na pracovní stanici)

Výhody:

- zjednodušení údržby a administrativy
- umístění tiskárny pod dohled

Je umístěn na známé adrese a uživatelé s ním mohou komunikovat zvláštním protokolem. Předávané pakety jsou dvojího druhu

- řídicí příkazy pro tiskový server (tiskni soubor, zruš tisk)
- vlastní data

Na požadavek tiskový server může reagovat

- odmítnutím požadavku

- zařazením požadavku do fronty a jeho pozdějším obsloužením

- akceptováním požadavku, informovat o tom vysílající uzel a začít s přenosem požadavku

Soubor příkazů tiskového serveru může zahrnovat např. tyto příkazy:

cancel - vypouští soubory z fronty a končí právě probíhající výstup

list - vypíše seznam souborů ve frontě serveru

add - přidá do tiskové fronty další soubor

data - přenos datových bloků souboru

Datové bloky jsou chápány jako část přenášeného souboru. Jsou buď ukládány, nebo okamžitě tisknuty. Neplatné nebo chybné bloky se odmítají. V závislosti na použitém protokolu si server s klientem vyměňují též potvrzení.

Výsledek požadované operace je indikován hlášením o výsledku operace (chybovým kódem).

K jednomu tiskovému serveru může být připojeno několik tiskáren, v síti může být instalováno několik serverů.

V běžných systémech (v OS klienta) je tisk souboru řízen driverem tiskárny. Zpřístupnění tiskárny prostřednictvím sítě znamená vyměnit tělo driveru a nahradit je novým tak, aby mohla být data posílána vzdálené tiskárně odpovídajícím protokolem přesměrování tisku.

Tisk souborů z klienta může probíhat

- bezprostředně - postupný tisk dat od jednoho klienta bez ukládání do souboru. Ostatní klienti jsou blokováni do ukončení tisku. Tisknoucí program musí ukončit tisk speciálním povelem - EOD (end of data - např. při uzavírání tiskárny nebo samostatný příkaz), který uvolní tiskárnu.

- zprostředkovaně - na serveru se vytváří z jednotlivých přenášených datových bloků soubor, který je po uzavření vytištěn. To umožňuje "souběžný tisk" několika klientů. Ukončení tisku je také indikováno příkazem EOD.

Poslední způsob využití tiskového serveru spočívá v tom, že v programu klienta přesměrujeme výstup z tiskárny do souboru, a ten pak předáme tiskovému serveru k vytištění.

2.2.5 Terminálový server (terminal server)

Běžným způsobem připojení terminálu k hostitelskému systému je použití přímých linek. Terminál však může být k hostitelskému systému připojen i použitím lokální sítě. Při přenosu sítí musí být data uzavřena do paketu síťového protokolu. Tyto funkce zabezpečuje terminálový koncentrátor (TC) nebo terminálový server (TS).

Terminálový koncentrátor je zařízení, k němuž je přímo připojen terminál. Přijímá znaky z terminálu, uzavírá je do paketu a posílá sítí do hostitelského systému. V hostitelském systému

se znaky rozbalí v driveru terminálu. Na obou stranách komunikačního řetězce je používán tentýž protokol.

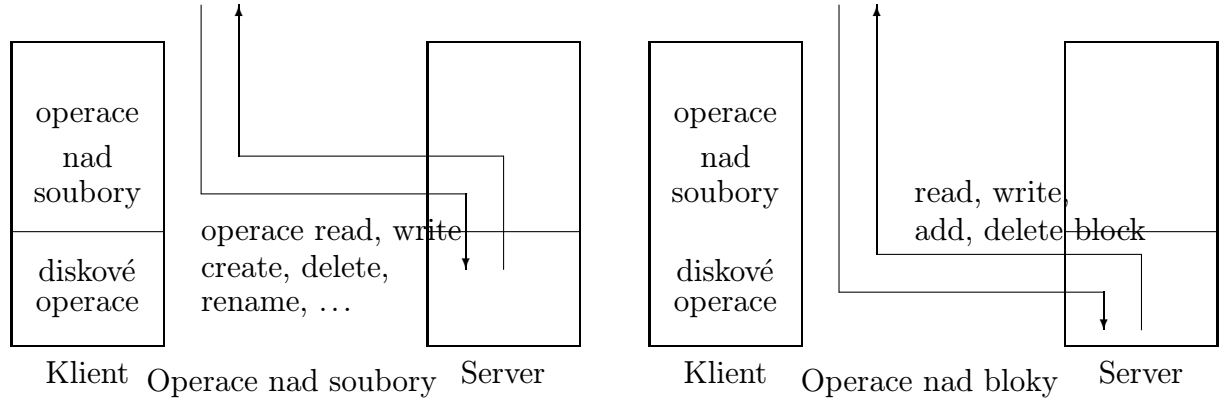
Průběh komunikace může být následující:

- terminál vyvolá pozornost TC (např. klávesou BREAK)
- TC se spojí s host. systémem a vytvoří virtuální okruh
- TC uvědomí terminál, že je navázáno spojení
- data z terminálu jsou předávána prostřednictvím TC do host. systému (a naopak)
- ukončení spojení provede uživatel přerušením TC.

Spojení terminálu s hostitelským systémem prostřednictvím lokální sítě představuje nejjednodušší případ. Komplikovanější situace nastane při propojení terminálu k libovolnému hostitelskému systému. K tomu se mohou použít koncentrátory, které multiplexují pakety (časově) a zabezpečují souběžné spojení několika terminálů s hostitelskými systémy (vytvořením několika virtuálních okruhů).

2.2.6 Diskový server (disk server)

Diskové jednotky jsou drahá zařízení, a proto se často požaduje jejich sdílení více uživateli. Podle úrovně programového vybavení, lze rozlišit dvojí způsob sdílení diskové paměti - na úrovni bloků disku nebo na úrovni souborů. Podle toho mluvíme o diskových nebo souborových serverech. Následující obrázek zachycuje jejich odlišnosti z hlediska operací, které zabezpečuje OS servera a které zabezpečuje OS klienta.



Obrázek 2.1: Souborový a diskový server

V této části se budeme zabývat diskovými servery.

Diskový server (DS) je v nejjednodušším případě stanice zapojená do sítě, mající k dispozici velký disk. Ostatní stanice mohou mít kromě lokálního disku ještě k dispozici přístup k tomuto společnému disku prostřednictvím diskového serveru.

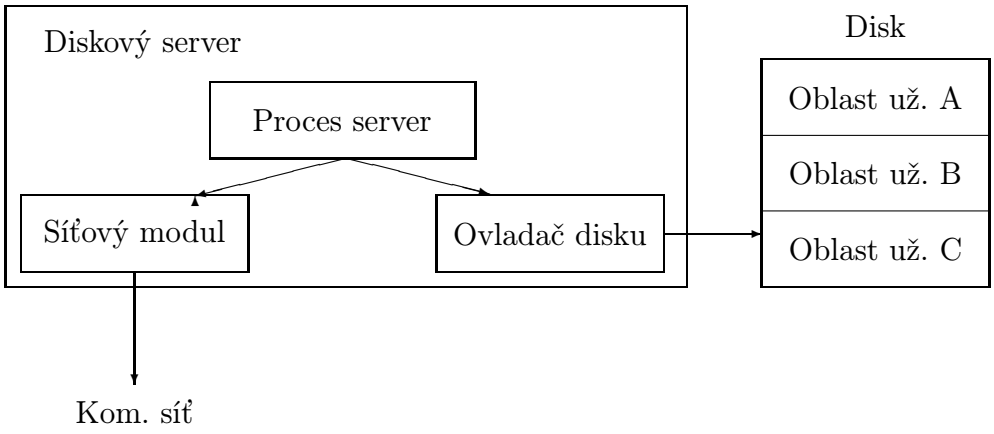
Realizace diskového serveru Realizaci diskového serveru lze rozdělit do dvou částí - na realizaci disku ve stanici klienta a realizaci disku ve vzdálené stanici servera.

Realizace disku ve stanici klienta Vychází se z principu realizace systému ovládání souborů (viz. Donovan Operační systémy). V lokálním uzlu se provedou operace pro převod čísla věty daného souboru přes adresu logické slabiky až na číslo bloku, počet bloků a posunutí v bloku. Podle čísla diskové jednotky se určí jedná-li se o lokální disk, nebo o disk vzdálený. V případě požadavku provedení operace na vzdáleném disku se výše uvedené parametry doplní o kód požadované operace (read, write, delete, add, ...) a číslo diskového svazku. Poté se přenosou pomocí komunikačního systému do uzlu servera jako "řídicí informace". Jedná-li se o operaci zápisu, přenesou se i zapisovaná data. Při operaci čtení naopak klient očekává příjem dat. Přenos dat může být tvořen i několika pakety (v závislosti na jejich objemu a typu použitého protokolu).

Jediným rozdílem mezi přístupem k lokálnímu disku a vzdálenému disku je použití komunikačních služeb. Z pohledu aplikačních programů uživatele se nic nemění.

Mezi procesem klienta a procesem serveru musí být vytvořen vhodný protokol (aplikační úroveň síťového programového vybavení).

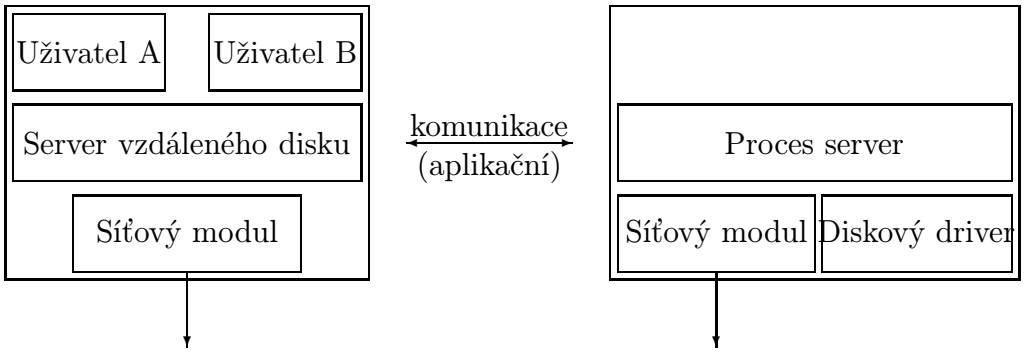
Realizace vzdáleného disku ve stanici servera Nyní se budeme zabývat otázkou realizace programového vybavení diskového serveru. Na disku serveru je paměť rozdělena mezi všechny uživatele. Programové vybavení obsahuje následující komponenty:



Obrázek 2.2: Diskový server

- **síťový modul** pro ovládání přístupu k síti s protokolem pro přenos dat
- **proces serveru**, který ovládá požadavky všech klientů. Klienti se chovají jako "modifikované" diskové ovladače a diskový server (proces) realizuje aktuální protokol klient - server.

Diskový server má k dispozici minimální soubor příkazů. Jsou to:



Obrázek 2.3: Uspořádání procesů diskového serveru

- otevření spojení (okruhu) - navázání spojení, vytvoření procesu serveru, kontrola přístupových práv a pod.
- ukončení spojení - zrušení spojení (okruhu), příp. procesu v serveru
- čtení bloku
- zápis bloku

Kromě služeb spočívajících v zabezpečení přenosu dat může server poskytovat klientovi i služby, související s ochranou přístupu. K ochraně může být použito hesla nebo přístupových práv. Podstatné je, že diskový server může zabezpečit pouze ochranu na úrovni celého disku (svazku), nebo na úrovni bloku (bez vazby na soubor nebo větu). Diskový server nezná ani jméno souboru, ani jeho uspořádání - pochopitelně obecně).

Každý klient může mít na sdíleném disku k dispozici vlastní vyhrazenou oblast, do které nemá nikdo jiný přístup. Server však může dovolit přístup i do sdílené oblasti (na sdíleném disku), kterou může využívat více klientů. Takováto oblast slouží k uložení sdílených programů (přístupová práva R/O).

Na závěr poznámka, týkající se homogenity použitých OS. OS klienta nemusí být totožný s OS serveru. Přístup k diskové oblasti je zabezpečen lokálním ovladačem disku. V případě použití homogenních systémů je možné oblast disku realizovat jako podstrom adresáře (a s celým tímto podstromem jsou spojena přístupová práva klientů - např. RWED). Oblast je "čitelná" i lokálním OS. Často se používá u síťových OS typu peer to peer (operační systémy si jsou z hlediska jejich postavení v síti sobě rovny, neexistuje nadřazenost a podřízenost jako v systému typu server/klient). Jsou-li OS nehomogenní (z hlediska organizace souborů), pak realizace diskového serveru znamená buď vyčlenění celého disku, nebo alespoň souboru (souborů) na disku. Tento soubor (tyto soubory) je pak chápán jako oblast. Struktura adresářů a uložení souborů pak odpovídá struktuře požadované OS klienta, nikoli OS servera. OS serveru tedy do vnitřního členění souboru nemusí vůbec "vidět" (a není to ani nutné). Na žádost poskytuje z oblasti pouze bloky oblasti (tedy z jeho pohledu věty jakéhosi souboru). K přístupu jsou použity standardní služby OS serveru.

Mnohdy se diskových serverů využívá k realizaci bezdiskových stanic klientů. Všechny přístupy na disk se provádějí prostřednictvím sítě (včetně bootování). Toto uspořádání má celou řadu výhod - jednodušší údržba programového i technického vybavení, cenově výhodnější, sdílení programových balíků, centralizované zálohování, zvyšování spolehlivosti za běhu zrcadlením nebo zdvojením disků, rychlé (mnohdy rychlejší než lokální disk - použití vyrovnávací paměti - cache). Mezi nevýhody patří jako výpadek celého systému při výpadku serveru, existence úzkých míst v komunikační síti a vlastním serveru.

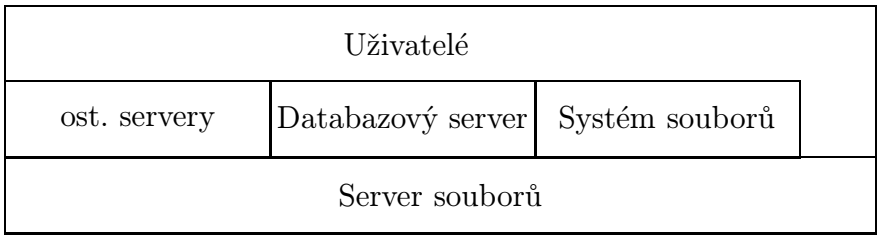
Modifikace OS klienta pro verzi s diskovým serverem znamená většinou pouze začlenění nového driveru, ostatní funkce OS se modifikují minimálně. Uživatelské programy se nemění.

Lze říci, že realizace diskového serveru je podstatně jednodušší, než realizace souborového serveru. Na druhé straně soubor služeb, poskytovaný souborovým serverem je daleko bohatší.

2.2.7 Souborový server

Souborový server (file server) FS patří k nejčastěji realizovaným komponentám distribuovaných systémů. Na rozdíl od diskového serveru podporuje operace nad soubory (např. create, erase, open, close, read, write), a dovoluje výhradní nebo sdílený přístup k datům různých délek od slabik přes věty až po celé soubory. Tento fakt dělá ze souborového serveru daleko výkonnější prostředek, než je diskový server.

Přístup uživatelů k serveru souborů je přímý, nebo prostřednictvím jiných serverů, jako je např. databázový server.



Obrázek 2.4: Vazba aplikačního programu na server souborů

Databázový server funguje buď jako server (z hlediska uživatelů), nebo jako klient s ohledem na server souborů. Představuje uživatelský databázový systém. Data jsou přístupná v základních jednotkách databázového systému - v rekordech. Uživatel zadává databázovému serveru příkazy pomocí jazyka (SQL).

Systém souborů představuje úplný systém souborů vytvořený nad serverem souborů. Dovoluje provádět operace nad strukturou adresářů a souborů, mechanismem ochrany souborů a pod. Nad serverem souborů může být klidně vytvořeno několik systémů souborů.

Kapitola 3

Realizace distribuovaných systémů

3.1 Předpoklady pro realizaci distribuovaných systémů

Zavedení decentralizace s sebou nese problémy, které je třeba eliminovat. Aby mělo smysl budovat decentralizovaný systém, je třeba řešit následující problémy:

Chyby Uzly v síti musí vykazovat nezávislé chybové režimy. Chyba v jednom uzlu nesmí ohroziit činnost jiného uzlu.

Jména Zdroje musí být jednoznačně označovány a lokalizovány. Jednoznačná lokalizace zdroje znamená, že musí být jednoznačně lokalizován v libovolném uzlu sítě.

Distribuované řízení Řízení je rozložené na jednotlivých uzlech sítě, neexistuje uzel, který by vykonával řízení centrálně.

Heterogenita Uzly sítě mohou tvořit počítače s různým způsobem zobrazení dat, různým instrukčním kódem a různou architekturou. Rovněž programové vybavení nemusí být homogenní, počítače mohou pracovat pod různými operačními systémy.

3.2 Způsoby budování distribuovaných systémů

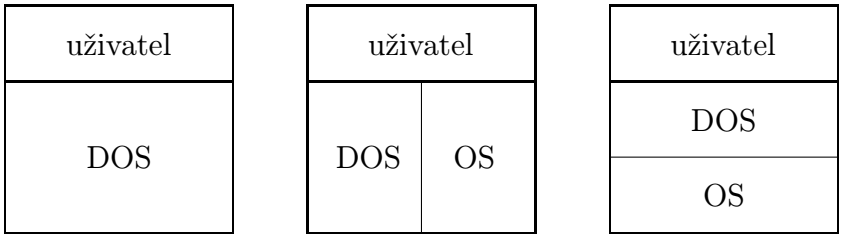
Budování programového vybavení jak v monoprocesorovém, tak i v distribuovaném prostředí je závislé na použitém operačním systému. Operační systém převádí symbolicky označované objekty na jejich konkrétní reprezentaci. Programátor proto mnohdy nemusí vědět, v jakém prostředí pracuje.

V distribuovaném prostředí se vytváří distribuované operační systémy. Jejich realizace i ladění je poněkud složitější, než je tomu tak u centralizovaných systémů. Distribuované operační systémy se vytváří jedním z následujících způsobů.

a) Od základů Vytváření od základů znamená, že vše bude vytvořeno znovu. Systém může být vytvořen na míru, což je výhodné. Objem programátorských prací je však značný. Obtížné a časově náročné je i ladění systému. Nedovuje přebírat zaběhnuté, odladěné programové produkty jak z oblasti systému, tak i z oblasti aplikačních programů.

b) Modifikací Modifikací existujícího operačního systému se minimalizuje množství programového vybavení, které je třeba vytvořit znovu. Většinou se jedná o některé funkce jádra systému. Během vývoje lze pro srovnání využít existující nedistribuované verze. Ladění je pak jednodušší. Mnohé práce mohou probíhat souběžně - na centralizované verzi lze ladit i modifikované decentralizované moduly.

c) Nadstavba Nadstavba nad existujícím systémem se používá tehdy, pokud nelze z nějakého důvodu zásadně modifikovat stávající systém. Realizuje se přidáním jedné nebo více vrstev, které pak pro uživatele vytvoří distribuované prostředí. Typickým příkladem je použití redirektoru v síťovém operačním systému jako je MS-DOS. Obdobně je vytvořen NFS (Network File System - systém pro sdílení souborů pro operační systém UNIX) i DCE (Distributed Computing Enviroment). V obou případech je operační systém schopen funkce jak v síťovém, tak i nesíťovém prostředí. Jádra operačního systému však musí být upraveno, aby mohlo funkce nadstavby akceptovat (např. funkce virtuálního systému souborů).



Obrázek 3.1: Úpravy centralizovaného operačního systému

V tomto obrázku zkratka DOS značí distribuovaný operační systém a OS (centralizovaný) operační systém.

3.3 Požadavky na distribuovaný operační systém

Oproti centralizovaným operačním systémům je třeba u distribuovaných nebo síťových operačních systémů zajistit podporu následujících funkcí:

Ochrana

Operační systém, pracující v distribuovaném prostředí musí zajistit ochranu svých zdrojů před neoprávněným přístupem uživatelů. Přibývají problémy s tím, že počítač je přístupný prostřednictvím sítě. Oprávnění uživatelé nejsou vázáni na vyhrazená pracoviště. Proti úmyslnému i neúmyslnému napadení se musí systém bránit pomocí speciálních prostředků, jako je ověřování uživatele a jeho práv, obranné valy a pod.

Duplicita

Zvýšené průchodnosti systému se dosahuje zvýšením počtu kritických zdrojů (zdvojení tiskáren, disků, vícenásobné kopie programů i dat). To s sebou nese problém souběžného přístupu k duplicitním zdrojům, přesměrování přístupu v případě chyby apod.

Transakce

Podpora transakcí dovoluje provádět efektivně souběžnou manipulaci se sdílenými daty.

3.4 Pojmenování objektů

Všechny objekty v systému musí být nějak označeny. K tomu se používá jednoznačný identifikátor objektu (někdy také nazývaný *binární jméno*). Tento identifikátor je přiřazen objektu vzájemně jednoznačně. Avšak binární hodnota, která je mu přiřazena, nedovoluje jeho jednoduché zapamatování. Proto se zavádí *symbolické jméno* objektu, které je voleno tak, aby s objektem nějak souviselo. Zavedení symbolického jména má však i další výhodu. Transformace symbolického jména na binární jméno nemusí být funkce. Jednomu jménu může být přiřazeno více (stejných) objektů. To pak umožňuje realizovat zálohování zdrojů, automatické přepnutí na ekvivalentní zdroj při výpadku apod. Obdobně více symbolickým jménům může být přiřazen jeden objekt.

Symbolické jméno může nebo nemusí být svázáno s umístěním objektu. Typickým příkladem takových objektů jsou soubory. Pokud symbolické jméno souboru, umístšného v hierarchii adresářů v sobě obsahuje též explicitně vyjádřenou cestu, pak je svázáno s umístěním objektu. Tato vazba se váže i k uzlu, na kterém je soubor umístěn. Typickým příkladem je označování souborů v operačním systému VMS fy DEC.

$$node\_name :: user\_disk[directory]file.extension; version$$

Výhoda takového označování spočívá v jednoduchém způsobu vyhledání místa uložení. Nevýhoda je v tom, že je jméno svázáno s konkrétním uzlem a adresářem. Při přemístění souboru na jiný počítač nebo do jiného adresáře se musí změnit i jeho jméno. Při kopírování souboru se také mění jeho jméno. Existují-li duplicitní kopie, má každá z nich jiné jméno, i když je obsah souboru identický. Opravovat se musí samostatně.

Obecnou příčinou tohoto jevu je to, že některé detaily nižší úrovně jsou vidět i na úrovni vyšší. Ukryjeme-li nižší úrovně, dosáhneme transparentnosti (transparentního pohledu).

3.5 Transparentnost

Při plné transparentnosti by se jednotlivé prvky systému jevily jednotně, bez vnitřní strukturovanosti. Dva studenti by nemohli vytvořit program stejného jména.

Pro dosažení úplné transparentnosti je třeba splnit následující podmínky:

a) umístění objektu nesmí být svázáno se symbolickým jménem objektu

b) symbolická jména musí být globálně jednoznačná

c) stejná jména musí mít stejný význam pro všechny.

Poslední podmínka se vždy nehodí. Např. pokud chceme, aby lokální (námi zavedená) jména měla přednost před jmény globálními. Dosažení úplné transparentnosti není vždy žádoucí a není mnohdy možné ji dosáhnout.

Existují požadavky uživatele, které kolidují s transparentností. Mezi ně patří:

a) lokální autonomnost Administrátor nebo vlastník uzlu očekává ponechání části lokální kontroly nad vlastními zdroji. V tomto případě autononost překrývá transparentnost.

b) optimalizace Další překážkou zavedení úplné transparentnosti je optimalizace využití vlastních zdrojů. Systém by měl dávat přednost zdrojům, které jsou z daného uzlu lépe přístupné, které nejsou přetížené a pod.

Úplné transparentnosti je obtížné dosáhnout i z důvodů odlišného technického a programového vybavení. Problém se řeší vytipováním omezené podmnožiny funkcí, které operační systém musí zabezpečovat ve všech uzlech.

3.6 Řízení

V lokálním uzlu je řízení velmi efektivní. V distribuovaném systému může být výsledek ovlivněn více uzly a vzdálenostmi mezi nimi. Algoritmy řízení jsou rozprostřené (distribuované) po celé síti.

Distribuované řízení se snažíme realizovat proto, abychom dosáhli:

- vyšší spolehlivosti
- vyšší dostupnosti
- vyšší výkonnosti.

Hlavním problémem distribuovaného systému je to, že neexistuje globální stav. U centralizovaných systémů lze stavovou informaci uložit do společné paměti, ke které mají přístup všechny procesory. V distribuovaném systému taková paměť neexistuje. Informaci o stavu jednotlivých komponent si musí uzly vyměňovat pomocí komunikačních funkcí. Vzhledem k rozlehlosti systému není možné zaručit dosažení takového stavu, aby pohled jednotlivých komponent na systém jako celek byl stále jednotný.

Decentralizace řízení musí právě tento rozpor překonat. To lze pouze tak, že jednotlivé uzly budou přísně synchronizovány. Na druhé straně přísná synchronizace vede k prodloužení doby provedení požadované akce. Proto se velmi často distribuované řízení realizuje tak, že v ustáleném stavu, kdy systém běží stabilně, je řízení prováděno centralizovaně (jedním uzlem nebo deterministickým předáváním řízení mezi uzly). Při poruše, kdy dojde k výpadku centrálního řízení se pomocí decentralizovaného algoritmu centralizované řízení obnoví.

Kapitola 4

Komunikační aspekty

Propojení jednotlivých uzlů v distribuovaném systému je realizováno pomocí komunikačních linek. Využívá se jak dvoubodových, tak i mnohabodových spojů. Propojení může být provedeno jak technologiemi, známými z lokálních sítí, tak i technologiemi MAN nebo WAN.

Komunikační programové vybavení je postaveno buď na vlastních základech, nebo využívá standardní protokoly známé z počítačových sítí. Vlastní komunikační programové vybavení se speciálními protokoly se používá zejména tehdy, je-li třeba pokrýt speciální požadavky (např. speciální potvrzování při skupinovém adresování, vyhodnocení výsledku přenosu bezprostředně po příjmu nebo během příjmu zprávy a pod.). Častěji se však používají standardní komunikační protokoly a sítě.

Pro uživatele je tato komunikační vrstva ukryta pomocí několika primitivních komunikačních funkcí s přesně specifikovanou sémantikou.

Pro komunikaci se používají dva základní prostředky - programová rozhraní:

- Systémy s posíláním zpráv
- Volání vzdálených podprogramů

Systém posílání zpráv vychází buď ze speciální množiny funkcí, nebo se používají modifikované funkce systému ovládání souborů. Volání vzdálených podprogramů je nadstavba nad systémem posílání zpráv. Jeho výhoda je v jednoduchosti, nevýhoda v omezenosti použití.

4.1 Posílání zpráv

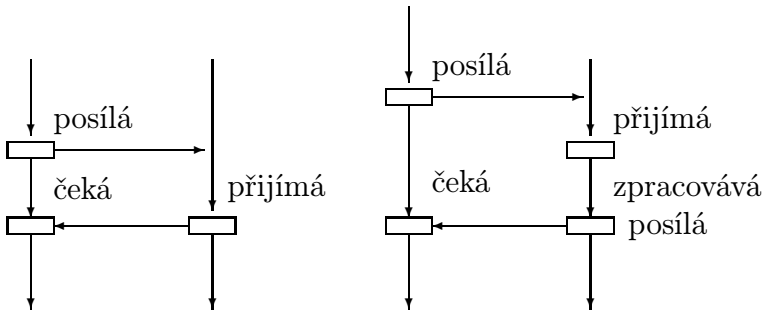
Posílání zpráv představuje jednu z možností jak komunikovat mezi procesy. Je reprezentováno bohatým výběrem funkcí, které je možné použít. Charakteristická je dobrá přizpůsobivost potřebám aplikací.

Systémů pro posílání zpráv mezi procesy bylo v minulosti vyvinuto hodně. Lze je dělit podle různých kritérií do několika skupin.

- **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 souběžně další operace (nezůstal zablokován čekáním na ukončení vyvolané operace).



Obrázek 4.1: Synchronní přenos zpráv

- **s využitím vyrovnávací paměti/bez využití vyrovnávací paměti**

  Vyrovnávací paměť, obsahující zprávu se může nacházet

  - v paměti odesílatele - odesílatel musí vyčlenit paměť a v případě asynchronních operací nesmí její obsah přepsat do té doby, dokud není zpráva přijata. Je vhodné pro synchronní přenosy.
  - v paměti komunikačního programového vybavení odesílatele - data jsou před odesláním přenesena do systémové vyrovnávací paměti. Původní oblast je možné okamžitě využít pro jiná data. Nevýhodou je nutnost rezervovat dostatečně (nepredikovatelně) velkou oblast systémové paměti.
  - v paměti komunikačního programového vybavení příjemce - data jsou přenesena do komunikačního uzlu příjemce. Nebezpečí spočívá v tom, že si je příjemce neodebere (ukončení procesu příjemce). Z tohoto důvodu musí mít komunikační programové vybavení pojistku - časové omezení pro držení dat.
  - v paměti příjemce - data jsou přesunuta do vyrovnávací paměti procesu příjemce. Problémy mohou nastat tehdy, jestliže příjemce nerezervoval dostatečně velkou vyrovnávací paměť.
  - někde v systému - tento způsob se běžně používá při přenosu elektronické pošty. Pro přenos zpráv mezi procesy se nepoužívá.

- **spolehlivý/nespolehlivý komunikační protokol**

  Mezi spolehlivé komunikační protokoly patří spojově orientované protokoly a spolehlivé datagramové protokoly. Použitý protokol ovlivňuje sémantiku volání, zejména způsob obnovy po chybě. Zajištění spolehlivé služby je vždy spojeno s větší režií na úrovni komunikačního programového vybavení. Výhodou jsou jednodušší aplikační programy. Nespolehlivé služby se používají tam, kde není problém operace zopakovat (např. při zjišťování času).

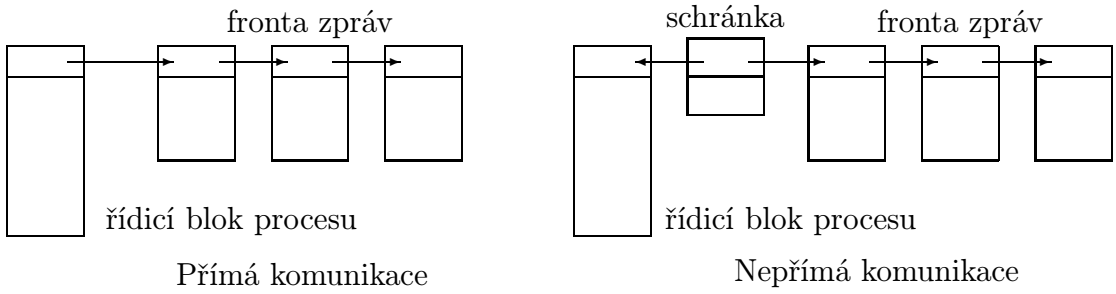
- **pevná/proměnná délka zprávy**

  Přenosy s pevnou délkou zprávy vedou k nutnosti zavézt fragmentaci dlouhých zpráv na úrovni aplikačního programu.

- **přímá/nepřímá komunikace**

  Přímou komunikací se rozumí komunikace mezi procesy, tj. cílová i zdrojová adresa je vztažena přímo k procesu. Např. je to adresa procesu (adresa nebo indentifikátor jeho řídicího bloku) nebo jméno procesu. Zprávy určené procesu jsou pak řazeny do fronty nebo front (fronta zpráv a odpovědí, fronta sdělení apod.).

  Při nepřímé komunikaci se používá pro označení odesílatele i adresáta číslo portu, tj. odkaz ne datovou strukturu, která může existovat nezávisle na procesu. S portem je pak spojena nějaká služba. Jeden proces může komunikovat prostřednictvím více portů, jeden port může obsluhovat více procesů.



Obrázek 4.2: Přímá a nepřímá komunikace

- **mapování adres**

  Mapování adres se provádí následujícími způsoby

  – zvláštním protokolem

    Příkladem je distribuovaný systém jmenných domén, používaný v Internetu k převodu jmen na adresy počítačů a naopak. Příkladem převodu jednoznačného identifikátoru skupiny programů na port je portmapper (RPC).

  – při překladu - tabulkou dvojic jméno - adresa. Nevýhodou je to, že se při jakékoliv změně v adresování musí znovu provézt překlad. Tento způsob je proto vhodný pouze pro stabilní převody.

- **randevous**

  Randevous je speciální případ komunikace mezi procesy. Využívá se k synchronnímu přenosu zpráv mezi procesy. Nevyužívá vyrovnávací paměti, data jsou přenášena přímo z paměti odesílatele do paměti příjemce. Pro procesy, které jsou umístěny ve společné paměti lze přenos dat uskutečnit pomocí mechanizmu segmentování nebo stránkování. Pro

komunikaci mezi procesy, umístěnými v různých uzlech systému lze použít jednoduchý komunikační systém.

Příkladem systému komunikujícího prostřednictvím posílání zpráv je programové vybavení (knihovny) operačního systému UNIX s rozhraním TLI nebo socketů. Podporovaná volání jsou si v obou uvedených systémech (a i v dalších, zde neuvedených) velmi podobná. Dále uvedené příklady se budou týkat pouze programového vybavení socketů. Poznamenejme, že výčet funkcí není zdaleka úplný.

Základní datovou strukturou, využívanou BSD sockety, je schránka. Komunikace je vždy svázána se schránkou, nikoliv přímo s procesem. Jedná se tedy o nepřímou komunikaci. Schránka obsahuje všechny informace, které s komunikací souvisí. Je to cílová a zdrojová adresa, cílový a zdrojový port, protokol a další údaje, související s rozpracovaností přenosů.

BSD sockety podporují datagramové služby (např. UDP) i virtuální okruhy (např. TCP). Kromě toho jsou podporovány i spolehlivé datagramové služby.

K vytvoření virtuálního okruhu se používá pasivní čekání (*listen*) a aktivní volání (*call*). Zprávy se přenáší pomocí funkcí *read* a *write*. K ukončení spojení slouží příkaz *close*.

Datagramové služby nevyžadují navázání spojení. Používají pouze příkazy *recv* pro příjem a *send* pro poslání dat.

Voláním vhodných funkcí lze dosáhnout jak synchronní, tak i asynchronní komunikace. Vyrovnávací paměti jsou uloženy v procesu odesílatele.

4.2 Volání vzdálených podprogramů

Základní myšlenkou volání vzdálených podprogramů (RPC - remote procedure call) je použití mechanismu volání podprogramů tak, jak je známo z mnoha programovacích jazyků pro sekvenční úlohy. V nejjednodušším případě je při RPC 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. Pozastavený volající proces je po přijetí zprávy a zpracování parametrů aktivován tak, jako by volal lokální proceduru.

RPC je prvotní komunikační mechanismus pro distribuované programy. Je jednoduchý, jednoduše pochopitelný, obecný a může být efektivně realizován. Využití RPC vede při konstrukci distribuovaných programů k abstrakci od komunikačních detailů, chyb přenosu a pod.

V ideálním případě by RPC mohlo mít stejnou sémantiku jak pro vzdálené, tak i pro lokální volání. Distribuované prostředí by pak neovlivnilo jeho funkční chování. V praxi je však tohoto stavu těžké dosáhnout. Jedním z důvodů je to, že v praxi je velmi obtížné odstínit chyby, vznikající při komunikaci dvou vzdálených systémů. Navíc odlišnosti datových typů a potřeba posílat je ve zprávách typicky vede k tomu, že sémantika přenosu parametrů je buď omezená, nebo odlišná od lokálního volání.

Systémy vzdáleného volání se navrhují závislé na konkrétním jazyku, nebo pro vícejazykové a více počítačové prostředí.

Jednoduchost činí RPC atraktivním jako primární mechanismus pro komunikaci v distribuovaných programech. Nabízené služby však nejsou pro některé aplikace adekvátní. Zvláště

tehdy, jestliže je RPC systém navržen na minimalizaci latence. V mnoha systémech je důležitější vysoká propustnost, nikoli minimální doba zpoždění přenosu.

4.2.1 Problémy

Při návrhu RPC systému je třeba diskutovat následující problémy.

- **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.

Existuje několik dalších realizačních problémů, jako např. minimalizace latence při volání, minimalizace režie při přepínání kontextu, předcházení drahým spojením. Tyto problémy však diskutovat nebudeme.

4.2.2 Nalezení adresy podprogramu

RPC, tak jako systém pro spojování programů v jednopočítačových systémech, musí obsahovat mechanismus pro spojení volajícího programu s volanou procedurou. Volající program obsahuje jméno volané procedury. Toto jméno musí být spojeno s aktuální hodnotou pro volanou proceduru. Navíc je důležité zkontrolovat, aby rozhraní (nebo typ) procedury volajícího a volaného si odpovídaly. Situace při volání vzdálené procedury je stejná jako při volání lokální procedury.

Spojování může být provedeno několika různými způsoby.

- **rozšířeným statickým linkerem**, umožňujícím umisťování programů ve více počítačovém systému
- **dynamickým linkerem**, který dovoluje spojit programy kdykoliv
- **spojování programů za běhu**, kdy program obsahuje proměnné typu procedura, a do těchto proměnných dosazuje hodnoty speciálním výpočtem (např. prohlížením tabulky jmen procedur)

4.2.3 Heterogenita a přenos paramtrů

Konkurentně pracující systémy jsou ukázkou systémů s vysokým stupněm heterogenity: do jedné sítě je připojeno mnoho různých typů počítačů, programy pracující na těchto počítačích jsou psány v různých programovacích jazycích. Navíc mohou na různých počítačích běžet různé operační systémy. Důležitým problémem je dovolit programům, pracujícím v heterogenním

prostředí tohoto typu, aby mohli komunikovat. Aby to bylo možné, je třeba definovat sémantiku komunikace jazykově a strojově nezávislým způsobem.

Mnoho RPC systémů dovoluje komunikaci mezi heterogenními programovými komponentami přepínáním statických deklarací rozhraní vzdálených procedur. Deklarace rozhraní slouží několika účelům:

- volající a volaný se musí shodovat v typech argumentů a výsledku procedury. Deklarace rozhraní dokumentuje tuto shodu. Některé systémy dovolují výjimky při vzdáleném volání. Tyto výjimky jsou také v rozhraní popsány.

- deklarace rozhraní slouží jako základ pro kontrolu typů, dovoluje volajícímu i tělu volané procedury kontrolu správnosti typů.

- reprezentace dat na obou koncích může být různá. Deklarace rozhraní může být použita jako základ pro automatické generování vhodného konverzního kódu.

V celé diskusi předpokládáme, že máme odpovídající popis datových typů. Protože různé programovací jazyky mají různé popisy datových typů, typy použité ve zprávách mohou být v zásadě odlišné od typů použitých lokálně v každém programu. Typy, použité v popisu rozhraní pro vzdálené procedury musí být definovány nezávisle na konkrétním programovacím jazyce.

Jsou-li typy, použité ve zprávách odlišné od lokálních typů použitých v programech, musíme mít k dispozici prostředky pro konverzi takových typů. I když jsou typy shodné (např. systém s jedním jazykem), mohou pro reprezentaci shodných datových typů používat různé komunikační moduly různou reprezentaci. Pak je také třeba nějaká forma konverze. Mnoho RPC systémů používá pro takové konverze automaticky generované spojky (stub).

Při vyvolání vzdálené procedury se nejprve vyvolá lokální spojka klienta pro tuto proceduru. Spojka klienta uloží odpovídající reprezentaci argumentů do zprávy (tento proces je obecně nazýván marshalling - seřazování) a inicializuje výměnu zpráv s počítačem serveru. Na serveru je zpráva zpracována spojkou servera, která použije příchozí zprávu k rekonstrukci argumentů. Ty pak předá požadované proceduře. Když se procedura ukončí, provede se opačný proces: spojka servera uspořádá výsledky a pošle je zpět do spojky klienta. Spojka klienta rekonstruuje výsledky a vrátí je volajícímu progra- mu. Spojka obsahuje všechny detaily konverze a komunikace. Volající volá v lokální proceduře spojku klienta a procedura servera je volána prostřednictvím lokálního volání ze spojky servera.

V mnoha systémech je spojka generována automaticky z popisu vzdáleného rozhraní. Obvykle je třeba, aby existoval pro každý jazyk zvláštní generátor. Jestliže se typy použité ve zprávách liší od typů, použitých v programech, musí generátor spojky určit z popisu rozhraní, které lokální typy mohou být ve spojce použity. Předpokládejme například, že vzdálené volání podprogramu má definováno rozhraní, umožňující použít jako argument pouze posloupnost celých čísel. V některých jazycích může mít spojka klienta jako argument pole, v jiných pouze odkaz na pole. Všechno je v pořádku, pokud je korespondence mezi typy ve zprávách a lokálními typy logicky jasná. Jestliže jazyk nemá žádný typ, který přirozeně koresponduje s typem dané zprávy, případně má-li jich několik, musí mít programátor k dispozici mechanismus, který dovolí předejít implicitní nastavení, zabudované ve spojce.

Problém konverze mezi různými reprezentacemi může být řešen několika způsoby:

- definovat standardní reprezentaci pro každý typ zpráv a požadovat od vysílače i přijímače aby provedl konverzi, jestliže jejich lokální reprezentace neodpovídá standardu. Tento postup je velmi efektivní, jestliže lokální reprezentace odpovídá reprezentaci zprávy. Jestliže však jsou reprezentace různé, může být režie spojená s konverzí významná. Např. jestliže vysílač i přijímač používají stejnou reprezentaci, která se však liší od reprezentace zprávy, provedou se dvě konverze, i když to nemusí být třeba. Obecně problém závisí na množství dat, přenášených každým voláním a na tom, jakou část z celkového času pro přenos spotřebuje konverze.

- druhou možností je přenést data ve tvaru, který je vlastní lokální reprezentaci vysílače. Na přijímači pak požadovat, aby provedl odpovídající konverzi pro svou lokální reprezentaci. Zde je však problém v tom, že by přijímač musel být připraven provádět jakoukoliv konverzi, která by se v systému mohla vyskytnout (spolu s problémem přidávání dalších systémů s novými jazyky).

Pro některé typické aplikace, jako např. server/klient, je výhodnější, aby se konverze prováděla na straně vysílače. Některé systémy řeší tímto způsobem odlišnosti na nízké úrovni (např. pořadí slabik), ale pro většinu datových typů používají standardní reprezentaci.

Dalším problémem je, jak reprezentovat argumenty obsahující odkazy (pointery). Mnoho systémů zakazuje používat některé datové typy jako argumenty nebo výsledky volání RPC (např. odkazy na proměnné, odkazy na formální funkce a procedury). Jiné systémy povolují jejich použití s určitými omezeními a jinak, než jak se používají lokálně.

Heterogenita se výrazně odráží při implementaci. Např. různé RPC systémy používají k implementaci totožných nebo podobných sémantik různé síťové protokoly.

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

Závěrem lze shrnout, že podprogram, který má být volán pomocí RPC je oproti "normálnímu" podprogramu omezen v následujících směrech:

- parametry se přenáší hodnotou

- v podprogramech nemohou být globální proměnné

- podprogramy nesmí mít vedlejší efekty

- jako parametry je vyskytují hodnoty

- jako parametry nelze přenášet odkazy na funkce nebo procedury

4.2.4 Transparentnost

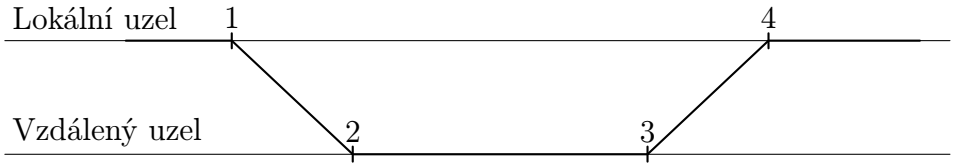
Transparentní pohled na volání vzdálených podprogramů omezují následující faktory:

- sémantika volání
- přenos parametrů
- zacházení s výjimkami

Sémantika volání

Základním cílem mnoha RPC systémů je vytvořit sémantiku volání pokud možno tak, aby se co nejméně lišila od sémantiky lokálního volání. Některým odlišnostem se však nelze vyhnout.

V prvé řadě ovlivňuje sémantiku volání možnost vzniku komunikačních chyb a chyb ve vzdáleném uzlu nebo místním uzlu.



Obrázek 4.3: Průběh volání vzdáleného podprogramu

Fáze 1-2 představuje přenos parametrů do vzdáleného uzlu, fáze 2-3 zpracování požadavku a fáze 3-4 přenos dat ze vzdáleného uzlu.

Různé systémy zpracovávají chyby různým způsobem, což vede k různým sémantikám volání. Nejslabší sémantiky nedávají při výskytu chyb žádné záruky. Vyvolání může být uskutečněno jednou, vícekrát nebo vůbec. Každé volání vzdálené procedury může být úspěšně ukončeno, nebo může skončit chybou někde uprostřed.

Dosažení silnější sémantiky může být složité. Nejsložitější jsou ta volání, která mají vnější efekty, způsobující některé akce, které se projeví mimo počítač.

Nastane-li chyba při vzdáleném volání, není jednoduché zjistit ve které fázi nastala. Obecně lze postupovat následujícím způsobem.

- **funkci vyvoláme pouze jednou**

  Nastane-li chyba, chybu nahlásíme nebo vyvoláme správce výjimek, který zjistí, ve které fázi chyba nastala a zjedná 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 servera 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.

Přenos parametrů

Druhou oblastí, kde je obtížné dosáhnout transparentnosti je přenos parametrů. Jak bylo diskutováno dříve, datové typy, použité ve zprávách se v heterogenních systémech od typů použitých v programech liší. Výsledkem je, že mnoho RPC systémů omezuje soubor datových typů, použitelných pro RPC. To vede k tomu, že většinou není možné nahradit bez úprav lokální volání procedury jejím vzdáleným voláním bez úprav. U SUN RPC je třeba navíc rozšířit počet parametrů pro vzdálené volání o deskriptor tohoto volání.

Zacházení s výjimkami

Poslední oblastí, ve které je těžké dosáhnout transparentnosti je zacházení s výjimkami. Různé jazyky obsahují různé mechanismy pro zacházení s výjimkami, od jednoduchého modelu ukončení programu, až po mnohem složitější modely obnovy.

Při vytváření vzdáleně volaných podprogramů nelze prakticky dosáhnout stavu, kdy by vzdálené volání přesně odpovídalo lokálnímu volání. Důvodem jsou zejména výše uvedená omezení při přenosu parametrů a dále chyby a jejich zpracování. Jedná se o:

- komunikační chyby
- chyby při přenosu parametrů
- zpracování výjimek při chybné činnosti podprogramu

4.2.5 Souběžné volání vzdálených podprogramů

Technika volání vzdálených podprogramů odpovídá lokálnímu volání i tím, že na straně klienta volajícího podprogram dojde k pozastavení činnosti do doby, dokud nedostane od servera odpověď. V jednodušších případech je možné obdobným způsobem realizovat i servera. Ten čeká na zavolání, provede podprogram a opět čeká. V mnoha případech však musí server zpracovávat požadavky od více klientů, které mohou volat servera souběžně. V tomto případě je nutné modifikovat spojku servera tak, aby mohl zpracovávat souběžně přicházející požadavky.

- postupné zpracování

  V místě servera se vytváří s použitím komunikačních prostředků fronta požadavků, které jsou pak serverem postupně zpracovávány. Problém souběžného příchodu požadavků je tedy řešen na úrovni komunikačního programového vybavení.

- vytvoření paralelního procesu

  Jedná se o oblíbenou techniku souběžného zpracování požadavků. Server na počátku vytvoří řídicí proces (master proces), který nedělá nic jiného, než že při příchodu požadavku vytvoří nový proces, svého potomka, který přebítá spojení s klientem. Tento proces slouží výhradně pro zpracování daného požadavku. Po ukončení zpracování požadavku proces zaniká.

Technika zpracování vzdáleného volání vytvořením paralelního procesu má výhodu zejména v možnosti souběžného zpracování požadavků při zachování jednoduchosti programu. Nevýhodou je značná režie spojená s vytvořením paralelního procesu a nemožnost jednoduché výměny dat mezi procesy (nutno použít systémové prostředky pro komunikaci mezi procesy - semafory, sdílenou paměť, roury a pod.

Modifikací výše uvedeného postupu je technika vytváření vláken (threads). Efekt je stejný, výhoda je v nižší režii spojené s vytvářením paralelní aktivity. Vlákna jsou součástí jednoho procesu. Přepínání běhu programu mezi vlákny může programátor ovlivnit. Navíc odpadá složitá komunikace mezi procesy - vlákna sdílí společnou paměť. Podprogramy spojené s funkcí vláken jsou součástí jádra operačního systému nebo jeho knihovny. Pro volané podprogramy platí omezení - musí být reentrantní.

Výše uvedená situace může nastat i v uzlu klienta. Souběžné provádění požadavků v uzlu klienta lze řešit obdobně. Má význam zejména v případě, kdy jeden uzel slouží současně jako server i jako klient. Další výhoda je v možnosti vícenásobného souběžného dotazu klienta.

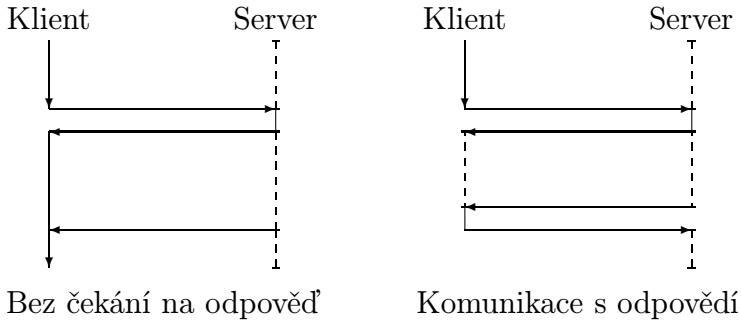
Protože ne všechny realizace RPC knihoven pracují na principu vláken, existují i další techniky, jak oprostit klienta od zahálivého čekání. Jednou z nich je *záměna volání klient/server*. Používají se dvě varianty:

- bez čekání na odpověď

  Klient vyvolá vzdálenou proceduru, ale nečeká na odpověď. Technicky to lze provézt např. nastavením časovače čekání na odpověď na nulovou hodnotu na straně klienta a neodesláním odpovědi na straně servera.

- komunikace s odpovědí

  Spočívá v tom, že odpověď je předána jako zvláštní volání ve směru server - klient (záměna rolí).



Obrázek 4.4: Záměna volání klient/server

4.2.6 Bezpečnost volání vzdálených podprogramů

RPC využívá běžné komunikační mechanizmy, a proto existuje nebezpečí napadení systému. Níže uvedené způsoby ochrany byly zavedeny pro SUN RPC.

- bez kontroly

  Klient se nemusí nijak autorizovat. Aplikace běžící na serveru může kontrolovat pouze adresu stanice klienta.

- heslem (odkrytý text) - community name

  Klient se autorizuje odkrytým heslem.

- číslo uživatele, číslo skupiny (UNIX)

  Autorizace klienta se provádí pomocí kontroly jeho UID a GID, které má jako uživatel nějakého operačního systému přiděleno.

- kryptografie (DES)

  Tímto způsobem lze zabezpečit buď celou zprávu (šifrovaný text), nebo zašifrovat zabezpečení textu (ochrana proti neautorizovaným změnám).

4.3 Spolehlivé broadcast protokoly

Základním problémem při návrhu distribuovaných systémů je vytvoření vhodného mechanizmu pro komunikaci mezi procesy. Po technické stránce se jedná o propojení uzlů na úrovni technického vybavení (např. spojení dvoubodovými spoji nebo mnohabodovými spoji lokálních počítačových sítí). Po programové stránce se jedná o přenos zpráv mezi procesy, zabezpečovaný buď spolehlivými virtuálními okruhy, nebo pomocí nespolehlivých datagramových služeb.

Vyšší formy abstrakce se snaží nezabývat se konkrétní formou přenosu a pohlíží na problematiku spíše z hlediska zabezpečení určitých cílů. Ideální představa vychází z úplného ukrytí distribuce před uživatelem. Umístění objektů se jeví jako by byly ve sdílené paměti.

Výhody této představy tkví v jednoduchosti programování, protože není rozdíl mezi psaním distribuovaných a nedistribuovaných programů.

Na druhé straně ale tato představa neodpovídá všem aplikacím. Některé musí mít explicitní znalost o umístění objektu, se kterým manipulují (např. z důvodu zabezpečení vyšší výkonnosti, dosažení odolnosti proti poruchám a pod.). Navíc implementace abstraktní sdílené paměti v síti počítačů může být extrémně neefektivní (zvláště pro velké sítě, kdy je těžké udržet režii abstrakce sdílené paměti, protože typické aplikace pracují pouze nad částí sítě).

Běžně používanou abstrakcí je vzdálené volání podprogramů (RPC). Zjednodušuje distribuované programování tím, že volání vzdáleného podprogramu vypadá takřka stejně jako jeho volání v lokálním uzlu.

RPC však může být použito ve většině případů pouze mezi dvěma procesy. Proto je vhodné pro model klient - server. Není vhodné zvláště v případech, kdy je distribuovaný program tvořen několika procesy s vysokým stupněm nezávislosti, a jestliže komunikace tuto nezávislost výrazně odráží. Typickým případem je implementace servera, který je z důvodu zvýšení odolnosti proti chybám implementován jako skupina procesů, rozmístěná v nezávislých uzlech. Tato komunikace má pak své specifické vlastnosti.

- klient nezná složení skupiny, a proto posílá své požadavky celé skupině najednou
- počet členů skupiny a jejich složení se mění v čase
- dělí-li se členové skupiny o práci podle požadavků, musí si být každý z nich jist, že jeho akce jsou konzistentní s těmi, které dělají ostatní členové skopiny.
- členové skupiny musí mít možnost komunikovat navzájem
- je třeba zajistit, aby proces mohl poslat zprávu skupině procesů. Takovému to vysílání se říká vysílání se skupinovou nebo všeobecnou adresu (broadcast).

4.3.1 Realizace vysílání se všeobecnou adresou

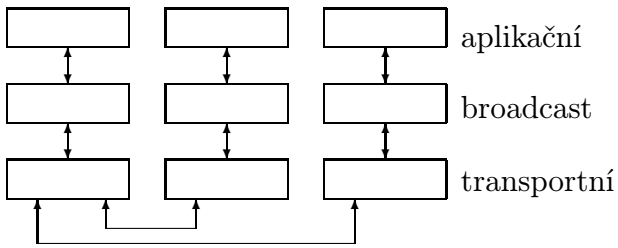
Vysílání se všeobecnou adresou se vyznačuje tím, že kopie zprávy je posílána všem cílovým procesům, zainteresovaným na všeobecném vysílání. Na rozdíl od spojení mezi dvěma uzly je třeba při vysílání řešit přípd, kdy některé procesy mohou skončit s chybou uprostřed vysílání. Zpráva pak není doručena všem cílovým uzlům, což se při komunikaci mezi dvěma uzly nikdy

nestane. Aby bylo vysílání se všeobecnou adresou použitelné pro programátora, musí se vytvořit dobře definované prostředí, které vliv chyby na aplikaci odstíní. Přenos zpráv, který tuto podmínku splňuje se nazývá **spolehlivé vysílání se všeobecnou adresou**.

Řešením je realizovat protokoly, které detekují chybu komunikace a spouští kompenzační akce. Příkladem takových protokolů jsou 2 fázové a 3 fázové protokoly pro realizaci transakcí.

Systémový model

Přenos zpráv mezi procesy s pomocí všeobecného vysílání je realizován v prostředí, kde neexistuje společná paměť. Procesy komunikují pouze prostřednictvím komunikační sítě. Předpokládejme, že zprávy jsou přenášeny pouze mezi dvěma libovolnými uzly sítě. Přenos zpráv je asynchronní, doba přenosu zprávy je proměnná.



Obrázek 4.5: Model subsystému pro vysílání se všeobecnou adresou

Komunikační subsystém pro vysílání se všeobecnou adresou je v tomto modelu řešen jako vrstva nad transportní úrovní, která transformuje jednu zprávu se všeobecnou adresou na N zpráv s individuálními adresami. Transportní služba doručuje zprávy spolehlivě a v pořadí, ve kterém byly odeslány. Tato forma spolehlivosti je dosažena použitím protokolů, které číslují zprávy a detekují ztracené nebo duplicitní zprávy.

Model chyb

V modelu komunikace se můžeme setkat s různým stupněm chybovosti uzlů.

- úplná chyba - jedná se o nejjednodušší model, kdy jediná chyba, která se může v systému objevit je výpadek procesoru a celého uzlu. Poškozené procesy nesmí vykonat chybnou akci.
- chyba vynechání - jedná se o chybu, kdy procesor náhodně ztratí přijatou zprávu. Je to realistická chyba ke které dojde např. při přetečení vyrovnávacích pamětí.
- byzardní chyba - jedná se o chybu, kdy si procesy vymýšlí a posílají falešné nebo odporující si zprávy.

Při vytváření protokolu je třeba předem stanovit, proti kterému typu chyby bude odolný. Model s úplnou chybou je nejjednodušší a hodí se např. pro automatické generování protokolu.

Chyba vynechání se musí řešit v komunikačních protokolech (např. číslováním paketů a zavedením časových omezení). Odstranění byzardních chyb je nejsložitější. Často je nutné zavézt kontrolu až na aplikační úrovni.

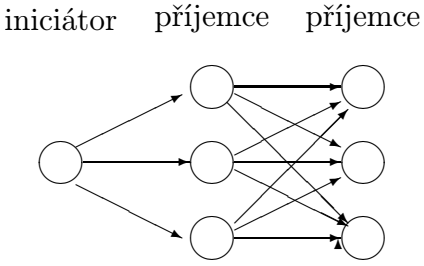
4.3.2 Atomický broadcast protokol

Jednou ze základních vlastností, kterou chceme spolehlivým broadcast protokolem dosáhnout je jeho atomicita. To znamená, že buď je zpráva přijata všemi fungujícími uzly nebo naopak žádným.

Model předpokládá pouze úplnou chybu uzlu. Nedoručení zprávy do uzlu se může objevit pouze tehdy, když uzel chybuje před ukončením protokolu.

Příkladem použití takového protokolu je oprava tabulky, jejíž kopie jsou rozmístěny v jednotlivých uzlech systému. Pokud se používá spolehlivý broadcast protokol, může uzel po přijetí zprávy opravit příslušnou položku v tabulce, protože obdobná zpráva byla určitě přijata i ostatními uzly. Pokud by nebyl použit spolehlivý protokol musel by uzel zajistit konzistentnost provedené operace jiným způsobem (např. použitím transakčního mechanizmu).

K zajištění spolehlivého broadcast protokolu však nestačí pouze spolehlivý transportní protokol. Během přenosů může dojít k chybě odesílatele i příjemce. Pokud je tato chyba uzlu totální (úplný výpadek), je možné situaci řešit pomocí jednoduchého atomického protokolu.



Obrázek 4.6: Jednoduchý atomický broadcast protokol

Proces iniciátor pošle zprávu m všem procesům. Příjemce přijme zprávu, a pokud nebyla předtím zpráva m přijata, pošle její kopii všem ostatním procesům. Jestliže uzel přijme zprávu a je funkční, obdrží od něj všechny uzly kopii. V konečné době obdrží každý z uzlů $N-1$ kopií zprávy m.

Uzel si musí pamatovat kopie všech přijatých zpráv do doby, dokud nejsou potvrzeny, nebo dokud si není jist, že jeho kopie byla doručena do všech uzlů.

Tento jednoduchý spolehlivý broadcast protokol má velké požadavky na komunikační cesty (každá zpráva je vysílána $(N-1)*(N-1)$ krát. Má velké požadavky na vnitřní paměť, protože si musí kopii zprávy pamatovat dostatečně dlouho. Má velké zpoždění, protože musí čekat, až dostane odpovědi ode všech zúčastněných uzlů.

Dalším problémem je typ chyby, kterou protokol odstraňuje. Jednoduchý atomický broadcast předpokládá, že uzel je buď funkční, nebo nereaguje vůbec.

Proto byly vyvinuty složitější protokoly, které výše uvedené nedostatky částečně eliminují.

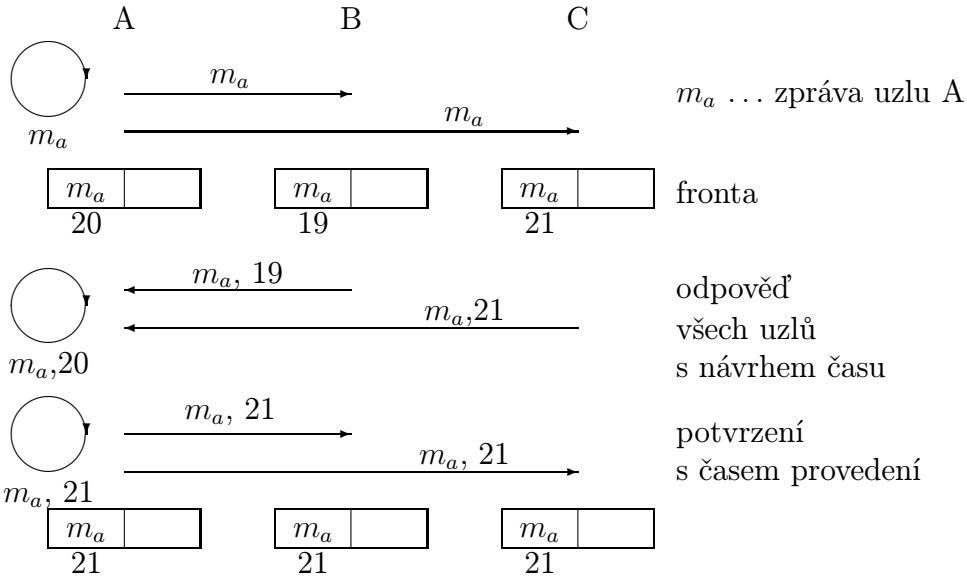
4.3.3 Uspořádaný broadcast protokol

Uspořádaný broadcast protokol nevyžaduje, aby byly operace v jednotlivých uzlech provedeny souběžně, ale aby byly provedeny ve stejném pořadí. Tato podmínka je sice slabší než ta, která je vyžadována u atomického broadcast protokolu, mnohým aplikacím však vyhovuje. Algoritmy, pracující na tomto principu je možné použít tam, kde je zcela decentralizován program pro manipulaci s daty, jejichž kopie jsou umístěny v jednotlivých uzlech. Jedná se o decentralizovaný přístup k mnohonásobným kopiím. Příkladem mohou být decentralizované zámky.

Zprávy jsou přenášeny do všech uzlů (zprávy se všeobecnou adresou). Součástí zprávy je časová značka, která rozhoduje o pořadí provedení příkazu. Čas může být reálný, nebo logický. Aby se jednoznačně určilo pořadí zpráv, je časový údaj doplněn o jednoznačný identifikátor uzlu. Ten se uplatní teprve v případě shodných časů.

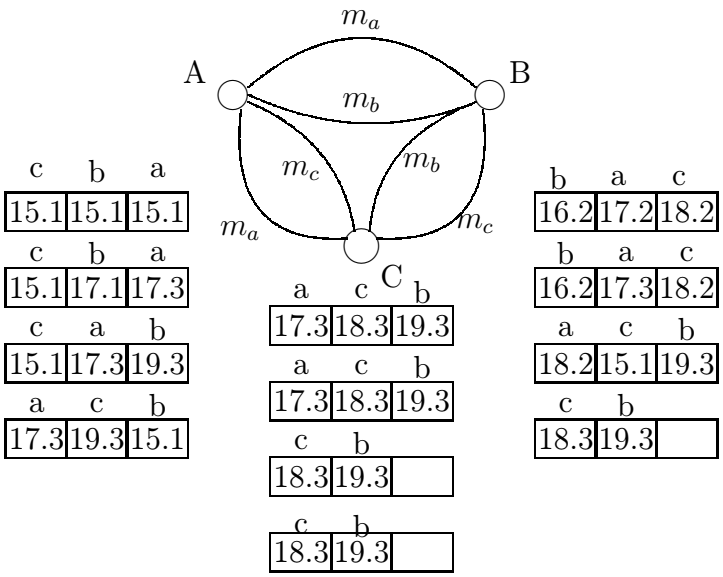
Protokol ABCAST

Protokol ABCAST pracuje následovně. Uzly vytváří skupinu. Uzel, který požaduje provedení operace budeme označovat iniciátor. Iniciátor vyšle zprávu se všeobecnou adresou do všech uzlů skupiny. Tato zpráva obsahuje identifikaci akce, která se má provézt. Všechny uzly skupiny (i iniciátor) odpoví zprávou s časovou značkou, která představuje návrh pořadí provedení akce. Uzel iniciátor vyhodnotí návrhy, vybere z nich nejvyšší hodnotu a odešle zprávu s touto hodnotou zpět všem členům skupiny. V jednotlivých uzlech se vytváří fronta požadavků, které se mají provézt. Fronta je uspořádána podle navrhovaného času provedení. K vykonání požadavku je třeba splnit dvě podmínky. Musí být nejstarší (na začátku fronty) a musí být potvrzen všemi členy skupiny. Situaci pro případ jediné zprávy ilustruje následující obrázek.



Obrázek 4.7: Stanovení času provedení v protokolu ABCAST

Obdobná situace by nastala i v případě vzniku více požadavků v jednotlivých uzlech.



Obrázek 4.8: Souběžné zpracování tří požadavků protokolem ABCAST

Výhodou výše uvedeného protokolu je to, že lze operace provádět souběžně se zachováním pořadí jejich provedení. Nevýhodou je problém udržení skupin. Výpadky uzlů skupiny je třeba řešit pomocí časových závislostí přímo v protokolu, nebo použít jiný, obecnější, protokol pro údržbu skupin.

Dojde-li k výpadku iniciátora, musí se po vyčerpání času požadavek zrušit. K výpadku člena skupiny může dojít před potvrzením, nebo po potvrzení požadavku. V obou případech se pokračuje. Ve druhém případě se nepozná, že člen ze skupiny vypadl.

Členství ve skupině se řeší speciálním protokolem. Výpadek člena znamená modifikaci tabulky členů. Přírůstek člena je signalizován všem členům skupiny. Nový člen skupiny obdrží informaci o členech, případně další informace (rozpracovanost dat).

K provedení jednoho příkazu je třeba $3 * (n - 1)$ zpráv. V každém uzlu se musí vytvořit fronta pro zapamatování zpráv, které nejsou potvrzené a nemohou se tudíž provézt (max. n zpráv).

Modifikace ABCAST protokolu

Předpokládejme, že členové skupiny tvoří logický nebo fyzický kruh. Iniciátor vlastní pověření, které je předáváno postupně všem členům skupiny. Provedení protokolu se pak zredukuje na dvě fáze. V první fázi (při prvním oběhu) pošle iniciátor zprávu všem členům skupiny. Spolu s touto zprávou pošle každý člen svoji časovou značku. Po návratu obíhající zprávy do uzlu iniciátora iniciátor vyhodnotí všechny časové značky a v druhé fázi pošle příkaz k provedení požadované operace spolu s výsledným časem.

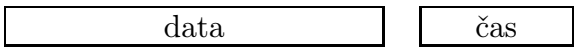
Další modifikací může být uspořádání členů skupiny do stromu. Iniciátor je kořenem stromu, zpráva se posílá od iniciátora do všech listů. Ke zkrácení doby odezvy dojde souběžným vy-

síláním zpráv po jednotlivých hranách stromu. Doba odezvy je dvojnásobek průměru grafu. Nevýhodou je nesnadná rekonfigurace při výpadku uzlu. Metoda je proto vhodná pro spolehlivé uzly.

Další modifikací může být síť se spolehlivými komunikacemi a uzly propojenými mnohabodovým spojem typu sběrnice. Použitím skupinového adresování můžeme redukovat nutný počet přenášených zpráv na $1 + (n - 1) + 1$, nebo dokonce na $1 + 1 + 1$.

Další modifikací může být přechod od pozitivního potvrzování k negativnímu. Uzel, který s navrhovaným časem nesouhlasí, pošle zprávu s vlastním navrhovaným časem. Ostatní neodpovídají. Po uplynutí stanovené doby iniciátor vyhodnotí odpovědi a pošle všem členům skupiny zprávu se stanoveným časem provedení.

Poslední zde uvedenou modifikací může být systém, který funkci majority realizuje již na úrovni rámce. Hodí se pro kruhové nebo sběrnicové sítě. Rámec se zprávou je doplněn o časové okno, obsahující navrhovaný čas provedení požadované operace. Všechny uzly dané skupiny mohou majoritní funkcí tento čas modifikovat bit po bitu (na úrovni komunikačního média). Výsledkem je hodnota, která odpovídá nejvyšší navržené hodnotě (viz prioritní metody řízení přístupu v lokálních počítačových sítích). V tomto případě stačí pro uspořádání jednoho požadavku jedna pouze zpráva.



Obrázek 4.9:

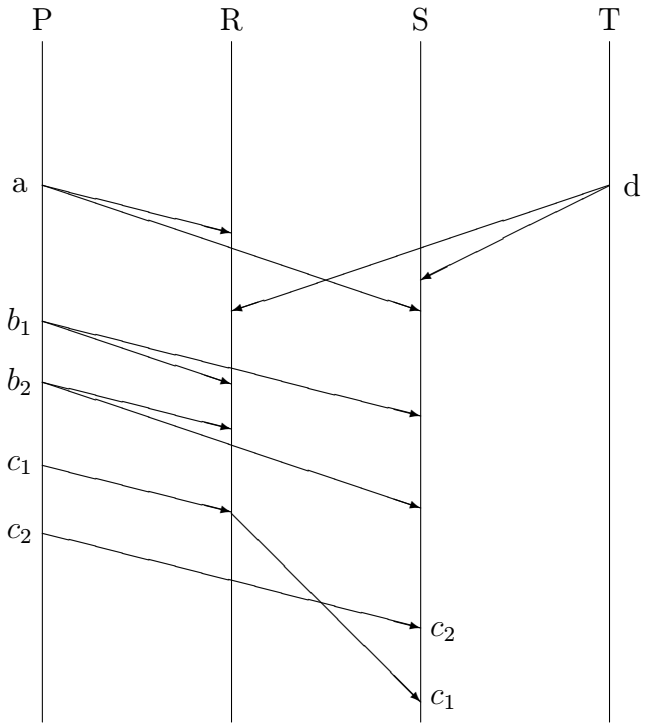
4.3.4 Protokol s oslabeným uspořádáním

V některých případech nevyžadujeme, aby se všechny události zpracovávaly ve všech uzlech systému ve stejném pořadí. Stačí nám, aby zprávy, generované v určitém pořadí jedním procesem byly zpracovávány procesy adresátů ve stejném pořadí, v jakém byly generovány. Přesněji můžeme požadavky formulovat následovně:

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 $a \rightarrow b$ a $b \rightarrow c$, pak je tato závislost tranzitivní ($a \rightarrow c$).

Princip řešení tohoto problému spočívá v přenosu všech zpráv, které byly příčinou události, která způsobila vyslání této zprávy. To pochopitelně není možné realizovat pro velký objem dat. Proto se celá historie většinou redukuje do pořadového čísla zprávy. V přijímacím uzlu jsou pak přicházející zprávy zachycovány do vyrovnávacích pamětí, řazeny a předávány ve správném pořadí přijímajícímu procesu.

Při použití cyklického číslování je třeba pouze dát pozor na to, aby v síti neexistovaly zbloudilé zprávy, jejichž číslo by se dalo zaměnit s aktuálním číslováním.



Obrázek 4.10: Příklady komunikací - protokol s oslabeným uspořádáním

4.3.5 Broadcast protokol pro dynamicky se měnící skupiny

Tento protokol se používá jako nadstavbový protokol nad jinými protokoly, které jsou používány pro manipulaci s vícenásobnými kopiemi, nebo při rozdělování výpočtu mezi více uzlů. Musí zajistit dynamický vzik 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.
3. Opustí-li proces skupinu, musí se o tom dovědět všichni ostatní členové skupiny. Za tímto účelem se používá speciální zpráva nebo protokol (např. ABCAST).
4. Provádí-li skupina operaci, musí být použity zámky.
5. Protokoly spolehlivého broadcastu mají přiřazeny priority v následujícím pořadí - ABcast, CBcast a GBcast.

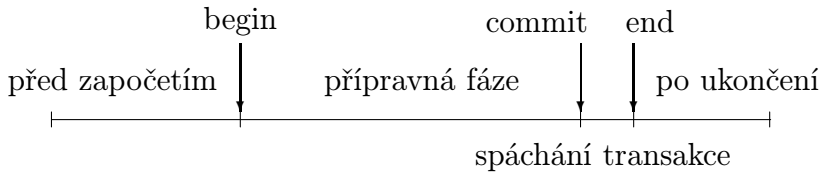
Kapitola 5

Transakce

Transakce jsou prostředek pro realizaci činností, které nelze od sebe oddělit. Název byl patrně odvozen ze stejnojmenných bankovních operací. Příkladem transakce je převod finanční částky z jednoho konta na druhé. Tato transakce nesmí být provedena pouze částečně. Po odečtení částky z jednoho konta se musí částka přičíst na druhé konto. Dojde-li během provádění transakce k chybě, musí být možné ji po odstranění chyby dokončit, nebo naopak musí být možné uvézt konta do stavu, ve kterém byla před započetím transakce.

Transakce probíhá ve dvou fázích.

- přípravná fáze, kdy se provádí příprava na modifikaci dat, provádějí se logické kontroly správnosti transakce. Všechny změny dat, chráněných transakcí jsou dočasné.
- fáze spáchání transakce, kdy se dočasné změny změní v trvalé

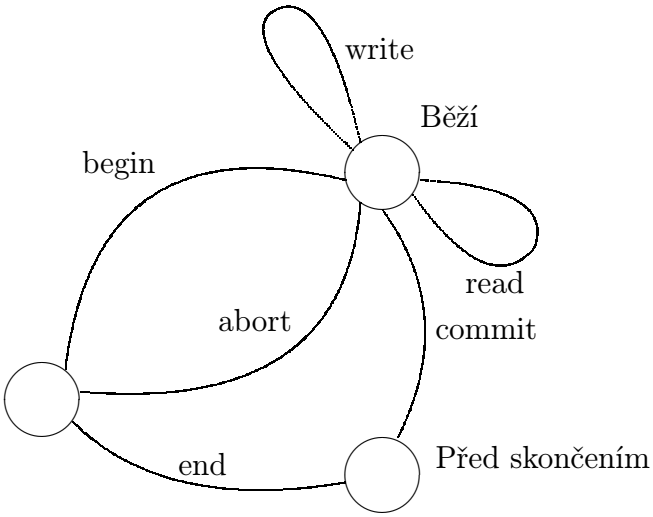


Obrázek 5.1: Časový průběh transakce

5.1 Vlastnosti transakcí

Obecně musí mít transakce následující vlastnosti

- Atomičnost
- Konsistentnost
- Isolovanost
- Trvalé provedení



Obrázek 5.2: Přechodový diagram transakce

Atomičnost Všechny výsledky výskytu chyby musí být odstraněny. Chyby transakce se nesmí projevit jinde.

Konsistentnost Konsistentnost transakce znamená, že vykonáním transakce přejde vždy databáze z jednoho konsistentního stavu do jiného konsistentního stavu.

Míru konsistentnosti transakcí je možné hodnotit podle způsobu manipulace s daty. Pojmem *pinav data* jsou označována taková data, která mohou být zapsána transakcí do databáze před spácháním transakce. Na základě koncepce špinavých dat lze transakce rozdělit do čtyř úrovní.

Stupeň 3: Transakce T je stupně 3 jestliže

1. T nepřepisuje špinavá data ostatních transakcí.
2. T neuskuteční žádný zápis, dokud nedokončí všechny své zápisy, tj. neprovede žádný zápis před spácháním celé transakce.
3. T nečte špinavá data jiných transakcí.
4. Ostatní transakce neušpiní žádná data čtená transakcí T dokud T neskončí.

Stupeň 2: Transakce T je stupně 2 jestliže

1. T nepřepisuje špinavá data ostatních transakcí.
2. T nespáchá žádný zápis před svým ukončením.
3. T nečte žádná špinavá data ostatních transakcí.

Stupeň 1: Transakce T je stupně 1 jestliže

1. T nepřepisuje špinavá data ostatních transakcí.

2. T nespáchá žádný zápis před svým ukončením.

Stupeň 0: Transakce T je stupně 0 jestliže

1. T nepřepisuje špinavá data ostatních transakcí.

Z hlediska stupňů konsistentnosti lze říci, že stupeň 0 brání pouze ztrátě změn, stupeň 2 předchází kaskádnímu abortu a stupeň 3 představuje úplnou isolaci, která zajistí, že kolidující transakce počká, dokud ostatní neskončí.

Isolovanost Isolovanost je vlastnost, která vyžaduje aby se každé transakci jevila databáze neustále konsistentní. Jinými slovy před ukončením transakce se její výsledky nesmí projevit ve výsledcích jiné, souběžně prováděné transakce. Čili nekompletní výsledky neukončené transakce nesmí být vidět jinou transakcí.

Trvalé provedení transakce Je-li jednou trasakce úspěšně ukončena, nemůže se stát, že by se její výsledky ztratily.

5.2 Operace nad transakcemi

Nad transakcemi jsou definovány následující operace

- **begin transaction**

  Tato operace představuje začátek transakce. Pro operace, které jsou prováděny uvnitř transakce platí zásady uvedené v předchozí kapitole.

- **end transaction**

  Znamená ukončení úseku programu provádějícího transakční operace. Po provedení této operace se buď provedly všechny operace bezchybně, nebo se neprovedla žádná (např. pokud došlo k chybě).

- **abort transaction**

  Operace se vyvolá tehdy, jestliže se zjistí, že transakce nemůže být úspěšně dokončena. Způsobí návrat všech změněných hodnot do původního stavu.

- **read**

  Operace čtení položky dat.

- **write**

  Operace zápisu dat.

- **commit transaction**

  Commit může znamenat v překladu *spáchat*. Tento poněkud neobvyklý překlad se nepoužívá, má však v sobě dynamiku, která se způsobem ukončení transakce zcela koresponduje. Proto jej budu používat. Commit je vnitřní funkce, která je vyvolána na konci transakce. Způsobí zápis všech změn do příslušných datových položek.

5.3 Vnořené transakce

Vnořené transakce jsou realizovány tak, že uvnitř transakce jsou volány další transakce. Tyto vnořené transakce lze pak spouštět 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 (abort).
- 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 dosáhne konce. Vše je pak vráceno do původního stavu (operace vycouvání - **undo**), všechny získané zámky jsou uvolněny nebo vráceny rodičům, a jejich rodiče jsou o tom uvědoměni. Rodič se může sám rozhodout, bude-li v transakci pokračovat, nebo svou transakci také zruší.

5.4 Prostředí pro realizaci transakcí

Vše, co bylo dosud napsáno vychází z předpokladu, že během provádění transakce nedojde k chybě. Připustíme-li výskyt chyb, můžeme je rozdělit do skupin podle jejich původu nebo závažnosti.

- **logická chyba** odhalená při výpočtu

  Výskyt tohoto typu chyb musí programátor předpokládat a program sestavit tak, aby byl schopen je odhalit a reagovat na ně. Jedná se např. o požadavek převodu částky z konta, přičemž se zjistí, že na kontě příslušná částka není. V tomto případě se po odpovídajícím testu vyvolá operace **abort** a transakce se zruší.

- **chyba technického vybavení**

  Výskyt tohoto typu chyb nemůže uživatel ani programátor ovlivnit. Mohou být způsobeny:

  - výpadkem procesoru
  - výpadkem paměti
  - chybou komunikačního média.

Oproti realizaci transakcí v centralizovaném prostředí přibyly chyby komunikačního média, které mohou způsobit stav,kdy nevíme, byla-li zadaná operace provedena či nikoliv. Podobné problémy nastanou dojde-li k výpadku některého z uzlů, realizujícího distribuovanou transakci. Pak v některých uzlech by mohla být provedena a v jiných ne.

Chyby lze dále dělit podle jejich závažnosti na:

- **trvalé/dočasné**

  Trvalé chyby se nejčastěji vyskytují u procesoru a paměti. Dočasné chyby jsou charakteristické pro komunikační médium a dají se odstranit opakováním požadované operace.

- **maskovatelné/nemaskovatelné**

  Maskovatelné chyby jsou takové, které může při jejich výskytu odstranit nižší vrstva programového vybavení. Typicky se jedná o komunikační chyby jako jsou duplicita přijatého rámce, chybějící rámec, změna pořadí přijatých rámců. Záleží na programovém vybavení, jak se s chybami tohoto typu vypořádá. Při použití spolehlivých služeb virtuálních okruhů se o nich aplikace nedoví. Při použití nespolehlivých služeb musí situaci napravit aplikace.

  Nemaskovatelné chyby nelze odstranit na nižších úrovních programového vybavení. Musí být zpracovány aplikací, ale nemusí vždy znamenat závažnou, neodstranitelnou chybu, která způsobí chybové ukončení celé aplikace. Jedná se např. o rozdělení sítě, uvíznutí procesů, časově neomezené čekání na zámek.

Chyby technického vybavení se mohou vyskytnout ve třech oblastech:

1. **Komunikační chyby**

   Komunikační chyby mohou být maskovatelné (duplikát, chybějící rámec), nebo nemaskovatelné (rozdělení sítí).Mohou být trvalé i dočasné. Při realizaci podpůrného programového vybavení je snahou zpracovat tyto chyby mimo aplikaci, v komunikačním nebo doplňujícím programovém vybavení.

2. **Chyby procesoru**

   Chyby procesoru mohou být objevitelné, které se zpracovávají jako výjimka, nebo nemaskovatelné, které však nezničí data. Nastane-li taková chyba procesoru, provede se nový start s následnou obnovou dat.

   Obnova dat se děje na základě informací, uložených na lokálních pamětech uzlu, případně získaných z jiných uzlů distribuovaného systému. V případě výpadku procesoru cizího uzlu se musí spustit algoritmy pro vypuštění tohoto uzlu (rekonfigurace).

3. **Chyby paměti**

   Chyby paměti jsou nemaskovatelné. Nastane-li chyba v paměti, je celistvost dat ohrožena nejvíce. Protože neexistuje ideální paměť, která by byla rychlá, levná, s neomezenou kapacitou a navíc si pamatovala i při výpadku napájecího napětí, používají se různé paměti, jejichž vlastnosti se doplňují. Z hlediska trvanlivosti uložených dat lze paměti rozdělit do následujících kategorií:

a) nestálá paměť

Nestálá paměť je paměť, která po výpadku napájení (nebo při jiné chybě technického vybavení) ztratí svůj obsah. Její výhodnou vlastností je krátká doba přístupu. Nejčastěji se vyskytuje jako vnitřní paměť počítače. Mezi tyto paměti patří všechny polovodičové paměti (SRAM, DRAM, kromě EEPROM).

b) stálá paměť

Stálá paměť se vyznačují tím, že po výpadku napájení udrží zaznamenanou informaci. Technická porucha však může způsobit ztrátu dat (např. zničená stopa pevného disku). Četnost takových chyb je však zanedbatelná. Stálé paměti se realizují jako polovodičové paměti se zálohovým zdrojem, elektricky programovatelné paměti (EEPROM), diskové paměti a pod. Mohou mít velkou kapacitu, doba přístupu k informaci je však většinou mnohonásobně delší než u pamětí předchozího typu.

c) trvalá paměť

Trvalá paměť se vyznačuje schopností udržet data za každých okolností. Odolává výpadkům napětí, poruchám technického vybavení i nosiče informací. Trvalá paměť je realizována zálohováním stálých pamětí, jako např. jejich zrcadlením, zdvojením nebo ostatními technikami používanými v diskových polích (RAID - Reliable Array Inexpensive Disks).

5.5 Funkční moduly potřebné pro realizaci transakcí

Realizaci transakcí můžeme rozdělit podle jejich funkce do několika modulů.

1. **Modul řízení transakcí**

   Slouží ke komunikaci mezi účastníky zpracování transakcí. Představuje koordinátora, který inicializuje transakci ve všech uzlech, ukončuje ji a potvrzuje její ukončení. Případně ji také ruší (abort).

2. **Modul řízení obnovy**

   Realizuje operace související s obnovením stavu po chybě. Operace REDO (dokonči) a UNDO (vycouvej) se provádí podle toho, v jakém okamžiku rozpracování došlo k chybě. Důležité je, aby ve všech zúčastněných uzlech byla transakce dokončena, nebo ve všech zrušena.

3. **Modul řízení vyrovnávacích pamětí**

   Vyrovnávací paměti se používají z důvodu snížení režie při přístupu do paměti. Data jsou ukládána do vyrovnávací paměti realizované jako nestálá paměť. Modul řídí přesuny do stálé nebo trvalé paměti.

4. **Modul řízení logu**

   Tento modul provádí zápisy do logového souboru. Logový soubor může obsahovat záznamy o hodnotách datových položek (stará hodnota, nová hodnota) - tzv. fyzický log, nebo popis jak provádět obnovu, tzv. logické logy.

5. **Modul řízení zámků**

   Použití zámků je jeden ze způsobů, jak řídit souběžný přístup k datovým položkám databáze, čili jak zabezpečit souběžné provádění transakcí. Zámky mohou být rozděleny podle typu na

   - zámky pro čtení, umožňující pouze čtení datové položky
   - zámky pro zápis, umožňující jak čtení, tak i zápis do zvolené datové položky
   - speciální zámky, spojené s realizací některých často se vyskytujících operací. Typickým příkladem je operace inkrementace.

   Uzamykat lze obecně různé objekty. V databázových systémech se jedná o

   - svazky
   - soubory
   - úseky v souborech (až na úroveň slabik).

6. **Modul řízení komunikací**

   Modul řízení komunikací slouží k zabezpečení výměny informací mezi jednotlivými uzly. Může obsahovat různé protokoly pro zajištění jednotlivých funkcí, charakteristických pro transakce (např. protokol pro ovládání zámků).

5.6 Obnova transakcí po chybě

Nastane-li při realizaci transakce chyba, je třeba dosáhnout její nápravy tak, aby byla zachována atomičnost transakce a zaručena stálost provedených změn. Proto bylo vyvinuto několik základních metod, které tyto podmínky splňují. Jedná se o:

- seznam požadavků
- stínové stránky
- záznam do souboru logů

Výše uvedené metody mohou být i vzájemně kombinovány.

5.6.1 Seznam požadavků

Souběžně s prováděním operací nad datovými stukturami databáze se provádí záznam o prováděných změnách do pomocného souboru. Tento soubor je umístěn v trvalé paměti. Záznamy obsahují informaci o tom, co se má udělat, když dojde k chybě během provádění některé z částí transakce. Záznamy do seznamu se řídí následujícími pravidly:

1. do seznamu požadavků se zapisují všechny změny dat. Záznam se provádí před změnou v databázi.

2. seznam požadavků je udržován v trvalé paměti. Informace se při výpadku nesmí ztratit.

3. manažer transakce kontroluje, byla-li transakce provedena

4. jestliže byla transakce provedena, změní se hodnoty v databázi a záznam ze seznamu se smaže

5. při chybě se zkontroluje, úplnnost seznamu. Je-li seznam úplný, nastala chyba až při zápisu do trvalé paměti. Proto může být zápis zopakován a poté ze seznamu vymazán. Není-li seznam úplný, nastala chyba před ukončením transakce Transakce se zruší a záznam se pak ze seznamu vymaže.

Při použití této metody musí být zaručeno, že je seznam úplný. Proto záznam o skončení transakce musí být do seznamu zaznamenán dříve, než je dán pokyn ke spáchání transakce. Zrušení transakce (abort) je velmi jednoduché. Objekty jsou opravovány až při ukončení transakce. Neukončenou transakci lze zrušit pouhým vymazáním záznamu ze seznamu. V případě poruchy serveru se po restartu prohlédne seznam požadavků, a podle rozpracování transakce se rozhodne, má-li být dokončena, nebo zrušena.

5.6.2 Stínové stránky

Základ této techniky spočívá v tom, že se nejprve vytvoří kopie originálu, a všechny změny se provádí do této kopie. Při ukončení transakce se změní pouze odkazy - nová stránka se zařadí na místo původní. Protože se jedná o jedinou, rychlou operaci předpokládá se, že při jejím provádění nenastane chyba.

Ve skutečnosti je paměť organizována jako strom (logické adresy nebo hierarchický systém souborů). Transakce vytváří nové hrany stromu zápisem nových spojů do nepoužitých míst v trvalé paměti. Spuštění transakce znamená vytvoření nové stínové stránky, spáchání transakce zařazení nové stránky na místo staré. Změny pokračují dokud není změněn společný vrchol. Tento vrchol je pak opraven po konzultaci s manažerem transakce. Poslední oprava se nazývá *Atomic Pointer Swap*. Transakce, která provede APS se ukončí. V případě, že dojde k chybě, po které následuje nový start systému, určuje APS zda-li se transakce zruší nebo ne.

5.6.3 Technika logů

Technika logů je nejznámnější technikou používanou pro obnovu transakcí po chybě. Logy jsou ukládány do souboru jako posloupnost záznamů proměnné délky. Soubor je udržován ve stálé nebo trvalé paměti.

Oprava objektu se provádí tak, že se objekt umístí do nestálé paměti a do logu se zapisují informace pro vycouvání (undo) zrušené transakce a dokončení (redo) ukončené transakce. Zápis do logu se musí provézt dříve, než se provede oprava objektu v nestálé paměti. Do logu jsou také zapisovány informace o ukončení nebo zrušení transakce. Po restartu jsou logy prohledávány a ukončené ale nezapsané transakce provedeny.

Logy se dají vytvářet dvěma způsoby. Podle obsahu je dělíme na:

- **fyzické logy**, které obsahují hodnoty proměnných spolu s jejich adresou umístění (stará hodnota, nová hodnota)
- **logické logy**, které obsahují příznaky rozpracování transakcí a seznam operací, které je třeba provézt při obnově transakce.

Do jednoho logu lze provádět záznamy pro více souběžně běžících transakcí. Logový soubor může být vyhrazen pro jednoho uživatele nebo pro více uživatelů. Může se také používat jediný společný logový soubor pro všechny uživatele. Změny lze provádět do vyrovnávacích pamětí (nestálé paměti) a zápisy do logu, umístěného ve stálé nebo trvalé paměti zapisovat po blocích. To má význam zejména pro zvýšení průchodnosti systému, protože zápis do nestálé paměti je rychlý a zápis do stálé paměti je možné provádět pouze tehdy, když dochází k ukončení transakce. Logy umožňují jednoduchý nový start systému. Stačí prohlédnout logový soubor a zpracovat nedokončené transakce. Logový soubor se prohlíží sekvenčně. To dovoluje vytvářet logy na médiích se sekvenčním přístupem.

5.7 Souběžné provádění transakcí

Pokud existuje v systému více procesů, které přistupují k objektům prostřednictvím transakcí, vzniká problém se souběžným přístupem ke sdíleným objektům. Při souběžném provádění transakcí je nutné zabezpečit bezchybnou manipulaci s daty. Existují tři známé způsoby řešení tohoto problému:

- postupné provádění transakcí
- serializace požadavků
- metody řízení souběhu.

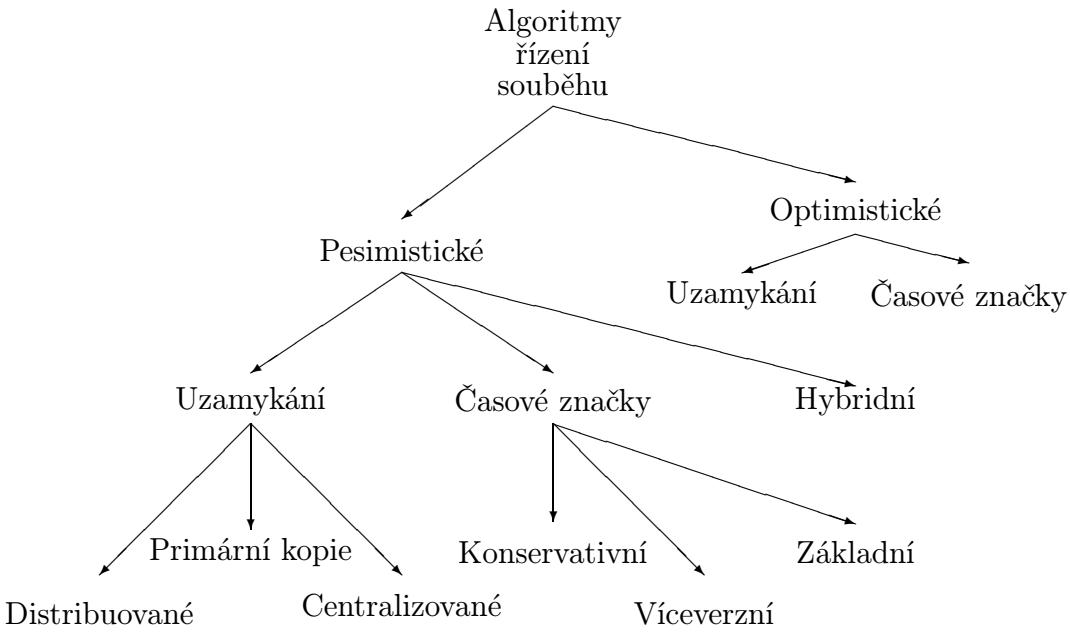
Nejjednodušším řešením je neprovádět transakce souběžně, ale postupně. Tato metoda se nazývá *postupné provádění transakcí*. Požadavky jsou řazeny do fronty a postupně, jeden po druhém, zpracovávány. Z hlediska správnosti nezáleží na pořadí, ve kterém jsou transakce provedeny. Tato metoda není efektivní, protože nedovolí provézt souběžně ani ty transakce, které se neovlivňují.

Metoda *serializace požadavků* vychází z předpokladu, že provádění několika transakcí souběžně může mít tentýž efekt jako nějaké postupné (nesouběžné) provádění. Takovéto uspořádání se nazývá serializovatelné. Jestliže souběžné vykonávání několika transakcí dává stejné výsledky jako nějaké postupné vykonávání, pak musí být také správné.

Avšak ne všechny požadavky lze serializovat, proto se používají metody *řízení souběhu*, kdy je průběh zpracování jednotlivých příkazů v transakci řízen tak, aby nedocházelo k chybné manipulaci s daty. Tyto metody vychází z techniky *zámků* nebo z techniky *časových značek* a mohou být *optimistické* nebo *pesimistické*.

Klasifikaci mechanizmů *řízení souběhu* lze souhrně vyjádřit následujícím schematem.

Metody řízení souběhu se dají dělit podle různých hledisek. Jedno z možných vychází ze způsobu synchronizace souběžně prováděných transakcí. Základní rozdělení vychází z postoje, jak problém konfliktu řešit.



Obrázek 5.3: Metody řízení souběhu

- **optimistické metody** předpokládají bezproblémový průběh transakce. Skutečný stav kontrolují až před jejím spácháním. Jestiže je vše v pořádku, pak se transakce provede (spáchá). V opačném případě se jedna z kolidujících transakcí zruší. Optimistické metody jsou vhodné tam, kde je pravděpodobnost konfliktu nízká.

- **pesimistické metody** se snaží předejít kolizím od začátku. Předem předpokládají možnost vzniku konfliktu a brání se mu jak mohou. To ale vede k příliš "opatrným" postupům, které mají významný vliv na průchodnost systému.

K uspořádání přístupu se u obou výše uvedených metod používá mechanizmus *uzamykání* a mechanizmus uspořádání podle *časových značek*. Obě metody je možné také kombinovat.

Uzamykání je často používaná metoda, vycházející z myšlenky povolit přístup jen jednomu. Další musí čekat na uvolnění. Podle způsobu řízení zámků lze dále uzamykání dělit na metody

- **centalizovaného uzamykání**, kdy jeden uzel je primární, obsahuje tabulky pro uzamykání objektů v celé databázi a rozhoduje o pořadí provádění požadavků, které generují ostatní uzly. Všichni se musí se svými požadavky obracet na primární uzel. Metoda je jednoduchá. Primární uzel však snižuje průchodnost a spolehlivost.

- **uzamykání primární kopie** spočívá v tom, že objekt, který můžeme uzamykat se v systém vyskytuje v několika kopiích. Jednu z těchto kopií označíme jako primární a spojíme s ní také zámek. Chce-li některý uzel provézt nad uvedeným objektem nějakou operaci, podmíněnou použitím zámku, musí o něj požádat uzel, ve kterém je uložena primární kopie. Manipulace s objektem se provádí lokálně. Před uvolněním zámku je nutné opravit všechny kopie.

- **decentralizované uzamykání** spočívá v tom, že řízení zámků je rozloženo po celé síti. Přístup k zámku znamená spustit decentralizovaný algoritmus, který zajistí koordinaci programů pro uzamykání, které běží na všech uzlech systému. Výhodou je to, že neexistuje centrální uzel pro uzamykání. Systém není citlivý na výpadek některého z uzlů. Může být odolný i proti rozdělení sítě na více částí. Nevýhodou je složitější algoritmus uzamykání, který musí v sobě zahrnovat i řešení výpadků uzlů nebo rozpad sítě.

Časové značky dovolují provádět operace nad objekty, umístěnými v různých uzlech, ve stejném pořadí. Časové značky generované jedním procesem tvoří monotóně rostoucí posloupnost hodnot. Jejich generování musí být organizováno tak, že se nikdy nesmí vyskytnout dvě stejné. Pokud dva procesy vygenerují stejnou hodnotu, uplatní se číslo procesu, které je vždy jedinečné.

Časové značky jsou generovány logickými hodinami, které vlastní každý proces. Pro řízení těchto hodin platí následující pravidla:

- přijme-li proces P_i od procesu P_j zprávu s časovou značkou h_j, nastaví logické hodiny na hodnotu
$$h_i = max(h_i, h_j) + 1$$

- posílá-li proces P_i zprávu, zvýší hodnotu h_i o 1

- mezi dvěma vnitřními událostmi může být hodnota hodin zvýšena kdykoliv, nemá to však význam.

5.7.1 Algoritmy založené na uzamykání

Základní myšlenkou řízení souběhu pomocí zámků je kontrolovat požadavky uživatelů na přístup k datům a povolit pokračování pouze těm, které nemohou v žádném případě kolidovat. Používá se technika *dvoufázového uzamykání*, kdy první fází je myšlena fáze požádání o prostředky a druhou fází fáze zpracování požadavku.

Předpokládáme-li existenci pouze dvou operací nad daty - čtení a zápis, můžeme vytvořit následující tabulku konfliktů.

	read	write
read	kompatibilní	konfliktní
write	konfliktní	konfliktní

Slučitelné jsou pouze vícenásobné operace čtení. Abychom tohoto faktu využili, vytvoříme dva typy zámků. Jeden pro čtení, druhý pro zápis. Někdy jsou označovány jako sdílený a výlučný. Je-li použit sdílený zámek, je možné bez čekání realizovat pouze další požadavek na sdílený zámek. Požadavek na výlučný zámek úlohu pozastaví.

Pozastavení procesů nebo úloh se děje většinou s využitím prostředků operačního systému - postavením do fronty. Při nevhodném řazení zámků mohou procesy uvíznout.

Včasné uvolnění zámků Včasné uvolnění zámků dovoluje zpřístupnit data ještě před ukončením transakce. Používá se pro zvýšení průchodnosti systému. S včasným uvolněním zámků pro čtení nejsou problémy. Pouze v případě, že se hodnoty čtou vícekrát, je třeba uvolnit zámek až po posledním čtení, aby se vždy četly stejné hodnoty. Včasné uvolnění zámku pro zápis může způsobit problémy tehdy, když po uvolnění zámku dojde ke zrušení transakce. Pak musí být zrušeny i souběžné transakce, které s uvolněnými daty pracovaly (čtení i zápis). Dochází k tzv. *kaskádnímu* zrušení transakce (kaskádní abort). Protože je tento způsob složitý a je s ním spojena velká režie, nepoužívá se.

Uzamykání podle typu Určitého zlepšení propustnosti můžeme dosáhnout vytipováním dalších operací, které by bylo možné uzamykat samostatně, a tak eliminovat "nesnášenlivost" operace zápisu s ostatními operacemi. Příkladem takové operace je např. operace inkrementace. Tabulka konfliktů má následující tvar.

	read	write	increment
read	kompatibilní	konfliktní	konfliktní
write	konfliktní	konfliktní	konfliktní
increment	konfliktní	konfliktní	kompatibilní

Operace čtení i operace inkrementace lze provádět souběžně. Provádění operace zápisu vede k vyloučení souběžné manipulace.

Pesimistické metody s uzamykáním Pesimistické metody s uzamykáním lze rozdělit dále na

- **centralizované**
  Zámek je umístěn pouze na jednom uzlu. Chce-li klient uzamknout objekt, požádá o uzamknutí centrální uzel. Rovněž odemknutí se děje na žádost. Nastane-li situace, kdy vyžadují klienti souběžný přístup, jsou jejich požadavky řazeny do fronty na straně centrálního uzlu. Další možností je požadavky na uzamčený objekt odmítat, což ale vede k zvýšení zatížení komunikačních cest. Centrální uzel distribuuje zámek mezi ostatní uzly. Uspořádání vede k přetížení komunikačních cest k uzlu. Pro případ výpadku centrálního uzlu je třeba zařadit mezi komunikační služby periodický test funkčnosti (který dále zatěžuje komunikační cesty). Výpadek centrálního uzlu vede ke krachu, ze kterého je možná obnova vybráním nového centrálního uzlu a zopakováním všech nevyřízených požadavků. Výpadek klienta lze ošetřit časovým omezením. Po vyčerpání přiděleného času se podřízenému uzlu zámek odebere.

- **primární kopie**
  zámky se distribuují i do ostatních uzlů. Distribuce však probíhá na pozadí, nejnovější stav zámků zná pouze primární uzel. Záložní kopie je možné použít při výpadku primárního uzlu k obnově stavu.

- **decentralizované**
  Decentralizované metody vyžadují využít nějakého distribuovaného algoritmu. Může to být metoda předávání pověření, metoda hlasování, a pod.

Optimistické metody uzamykání

5.7.2 Algoritmy založené na časových značkách

Pesimistické metody využívající časových značek Ke každému objektu se vytvoří časová značka pro zápis (WTS) a časová značka pro čtení (RTS).

Pro čtení musí platit: $TS \leq WTS$

Pro zápis musí platit: $TS \leq RTS \wedge TS \leq WTS$

Optimistické metody využívající časových značek

5.8 Ukončování transakcí

Při realizaci vícenásobného uzamykání objektů je využíván algoritmus postupného uzamykání. Postupné přidělování zdrojů dovoluje lépe využívat zdroje a zvyšuje propustnost systému. Pro uvolňování zámků platí následující pravidla.

- **postupné uvolňování zámků** se děje v pořadí ve kterém byly přidělovány. Důvodem je zabránění vzniku uvíznutí. Metoda má výhodu v tom, že pokud proces nepotřebuje zámek, okamžitě jej uvolní pro další proces. Nevýhoda spočívá v tom, že dojde-li k chybě v transakci po uvolnění některých zámků, musí se opravit i ty transakce, které mezitím opravenou hodnotu požily. Ty mohou být ještě otevřené, ale také již uzavřené. V důsledku to vede ke zřetězení oprav a obecné řešení je složité.

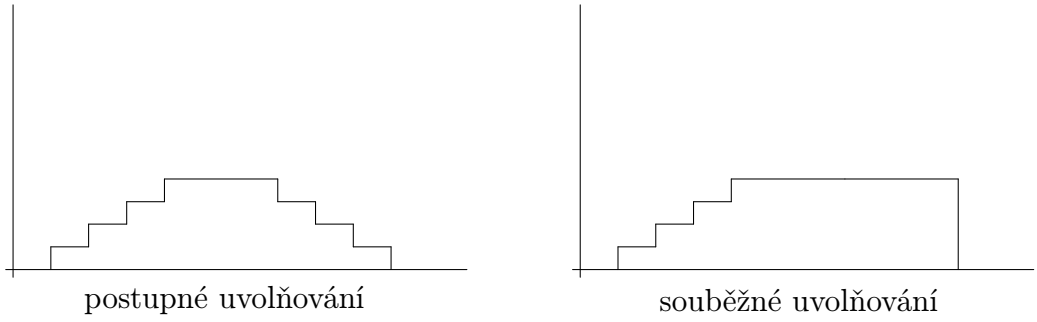
- **souběžné uvolnění zámků** odstraňuje předchozí nedostatky. Zámky jsou uvolněny teprve v okamžiku, kdy je již jasné, jak provedení transakce dopadne. Odpadají tedy problémy spojené s obnovou ostatních transakcí.

5.9 Distribuované transakce

Jsou-li datové soubory umístěny v jednom uzlu, je ukončení (spáchání) 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 je to dvoufázové nebo třífázové ukončování.

Při realizaci algoritmů pro distribuované transakce je třeba řešit následující problémy:

- rušení (abort) transakce, které může nastat v libovolném uzlu



Obrázek 5.4: Ukončování transakcí

- chybu technického vybavení libovolného uzlu nebo chybu komunikační sítě.

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 třífázového uzamykání.

5.9.1 Metoda dvoufázového ukončování

Pro transakce, jichž se účastní více než jeden server, musí být zvolen koordinační server. Je to často server, kterému je určen počáteční požadavek begin transaction. Ostatní servery vystupují jako cohorts (sluhové), protože musí poslouchat koordinátora.

K realizaci distribuované transakce vykoná server koordinátora následující kroky

- 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 servera.

Několik poznámek, jak se musí chovat koordinátor a sluha.

- jakýkoliv sluha je může kdykoliv zrušit nebo pokusně spáchat transakci dokud to neoznámí koordinátorovi. Od tohoto okamžiku musí poslouchat koordinátora a své rozhodnutí nesmí změnit.

- očekává-li koordinátor odpověď na svou výzvu, musí čekat dokud nedostane odpovědi ode všech sluhů. Protože v tuto dobu může dojít k chybě komunikace, čeká po časově omezenou periodu (časovač). Vyprší-li čas, považuje chybějící odpovědi za negativní a transakce se musí zrušit

- jestliže sluha skončí chybou po nastavení stavu podmíněné spáchání, ale předtím dostal zprávu commit nebo abort od koordinátora, musí být schopen ukončit transakci po obnově své činnosti. Jenže sluha neví, jak transakce nakonec skončila. Proto musí mít prostředky pro to aby zjistil, jak transakce skončila. Proto musí koordinátor uschovávat své záznamy do doby, dokud si není jist, že všichni sluhové transakci ukončili.

- jestliže zkrachuje koordinátor, musí to být sluhy detekováno, aby mohli zvolit nového koordinátora

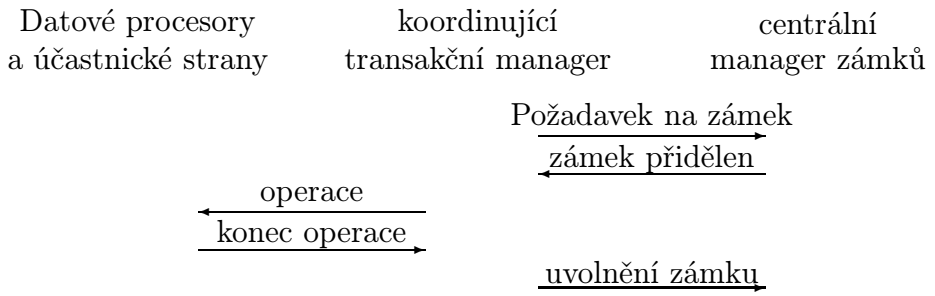
- tento protokol je velmi náročný z hlediska počtu vyměňovaných zpráv. Existují již optimalizace tohoto algoritmu.

- dvoufázové ukončení transakce je velmi starý a široce používaný protokol

Podle způsobu komunikace lze rozdělit dvoufázové ukončování distribuovaných transakcí do tří skupin:

- centralizované
- lineární
- distribuované

Centralizované dvoufázové ukončení transakce Provádění transakcí v distribuovaném prostředí předpokládá existenci koordinačního servera, který transakci inicializuje a řídí její provedení a ukončení. Dvoufázové ukončování transakce spočívá v tom, že v prvé fázi tento koordinační server inicializuje transakci a vyzve ostatní uzly (často označované jako sluhové) k jejímu provedení. V druhé fázi čeká na odpovědi. Jsou-li všechny kladné, vydá příkaz ke spáchání transakcí 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 transakcí z důvodu synchronizace.



Obrázek 5.5: Centralizované dvoufázové ukončení transakce

Lineární dvoufázové ukončení transakce V případě lineárního ukončování transakce je provádění jednotlivých transakcí v jednotlivých uzlech zřetězeno. Koordinační server vydá příkaz k inicializaci transakce v prvním uzlu, ten provede příslušné akce první fáze a předá 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 má-li se transakce provézt. Pokud ano, pak se provede (spáchá) a předchozímu uzlu se předá příkaz ke spáchání transakce. Nemá-li se transakce provézt, předá se příkaz ke zrušení. Zřetězením v opačném směru projdou příkazy spáchej transakci nebo zruš transakci až ke koordinačnímu serveru. Ten provede/zruší transakci jako poslední.

Distribuované dvoufázové ukončení transakce Distribuované dvoufázové 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.

Kapitola 6

Synchronizace v distribuovaných systémech

Metody synchronizace v distribuovaných systémech můžeme rozdělit do dvou skupin podle způsobu řízení.

- centralizované
- decentralizované

Máme-li zajistit synchronizaci dvou a více činností, máme obecně k dispozici následující prostředky

- zámky
- časové značky
- předávání pověření
- čítače událostí
- sequencery

Synchronizace ve svém důsledku znamená omezovat některý z procesů v jeho činnosti. Obdobně jako v případě transakcí existují dvě metody

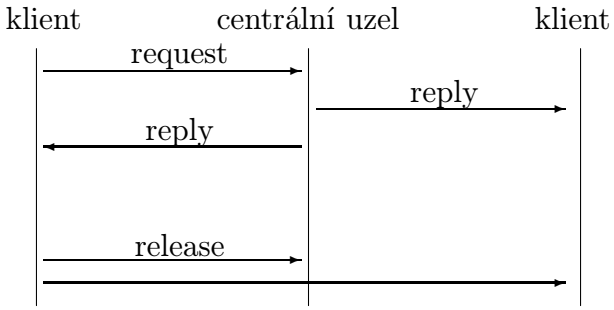
- optimistické metody
- pesimistické metody

6.1 Centralizované mechanizmy synchronizace

Centralizované mechanizmy synchronizace předpokládají existenci centrálního uzlu nebo procesu, který synchronizaci zajišťuje na základě požadavků ostatních. Výhodou tohoto řešení je jeho jednoduchost, nevýhodou výpadek celého systému při výpadku centrálního uzlu a zvýšené zatížení komunikačních cest k tomuto uzlu. Proti výpadku centrálního uzlu se lze bránit výběrem nejspolehlivějšího uzlu, nebo dynamickým výběrem některého z uzlů. Při jeho výpadku se např. pomocí algoritmu výběru jednoho z N vybere nový centrální uzel.

Centrální fyzické hodiny Synchronizace spočívá v tom, že existuje uzel, mající k dispozici časovač. Tento časovač určuje čas systému. Z časovače jsou periodicky rozesílány zprávy jednotlivým uzlům. Ty se podle doručeného času synchronizují. Nevýhoda spočívá v tom, že doba doručení zprávy je závislá na náhodném zpoždění komunikační linky.

Fronta Synchronizace je zajištěna třemi zprávami (**request**, **reply** a **release**). Všechny požadavky jsou přenášeny do centrálního uzlu, který je staví do fronty. K provedení požadavku dává pokyn centrální uzel v okamžiku, kdy se požadavek objeví na začátku fronty. Při výpadku centrálního uzlu je nutné určit nový uzel (algoritmus 1 z N). Poté musí dojít k rekonstrukci fronty.



Obrázek 6.1: Synchronizace pomocí fronty požadavků

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

- **advance**, která nedělitelně zvyšuje hodnotu proměnné o jedničku

- **read**, která čte hodnotu proměnné

- **await**, která čeká, pokud je uvedená hodnota výrazu menší než hodnota čítače událostí.

Čítače událostí jsou synchronizačním mechanizmem, který je rovnocenný semaforům nebo kritickým oblastem. Má výhodu v jednoduché realizovatelnosti v prostředí, kde chybí sdílená paměť.

Aparát čítačů událostí bývá doplněn o proměnnou typu sequencer. Tato proměnná má poněkud jiný význam než jak je uvedeno v následujícím paragrafu. Jedná se o celočíselnou proměnnou, která nabývá nezáporných hodnot. Je nad ní definována operace **ticket**, která vrátí starou hodnotu sequenceru a současně zvýší hodnotu sequenceru o jedničku. Spolu s čítači událostí vytváří efektivní aparát pro realizaci synchronizace v distribuovaném prostředí.

Sequencery Sequencer je chápán jako uzel, který zajistí synchronizaci požadavků ostatních uzlů. Existují dva způsoby synchronizace.

V prvním z nich přijímá sequencer požadavky ostatních uzlů (např. pomocí RPC nebo jiného způsobu komunikace typu 1:1), řadí je do fronty a přiřazuje jim časová razítka. Je-li požadavek na řadě, pošle jej s pomocí skupinové komunikace ostatním uzlům ke zpracování. Přijetí požadavku si ověřuje na základě potvrzení, které by měl ode všech uzlů dostat. Aby byl algoritmus odolný proti chybám, pamatuje si sequencer i staré zprávy, aby mohl jejich vysílání případně zopakovat. Každý uzel si pamatuje časové razítko posledně přijaté zprávy od sequenceru. Jestliže nesouhlasí pořadí, požádá sám sequencera o zopakování chybějících zpráv.

V druhém případě posílá uzel požadavek všem uzlům včetně sequenceru. Požadavek je zapamatován v jednotlivých uzlech ve frontě. Sequencer také řadí požadavek do fronty. Přijde-li na něj řada, vyšle ostatním uzlům příkaz na provedení požadavku s odpovídajícím časovým razítkem. Vlastní požadavek se již neposílá.

Problém nastane při výpadku sequencera, kdy se musí najít nový sequencer a zrekonstruovat ztracená fronta.

6.2 Decentralizované mechanizmy synchronizace

Vícenásobné fyzické hodiny Vícenásobné fyzické hodiny pracují tak, že si vzájemně korigují lokální čas.

Vícenásobné logické hodiny Vícenásobné logické hodiny nedávají k dispozici reálný čas, ale obsah čítače, který se při každé události inkrementuje. Dostane-li uzel zprávu, opraví si hodnotu vlastního čítače tak, aby byla maximem z obou (vlastní i přijaté) hodnot, zvýšené o jedničku. Též při vlastních aktivitách hodnotu lokálního čítače zvyšuje.

Metoda obíhajícího příznaku Metoda obíhajícího příznaku spočívá v tom, že ten, kdo příznak vlastní může provádět požadovanou operaci. Další požadavky se umisťují do příznaku. Podle priority požadavků, zapsaných v příznaku se rozhoduje o jeho dalším vlastníku.

Kapitola 7

Algoritmy pro synchronizaci hodin

K šíření přesného času v počítačové síti slouží časové servery. Ty získávají přesný čas od časových etalonů. Vzhledem k tomu, že se požadavky na přesnost určení času neustále zvyšují, uplatňuje se při jeho určení i časové zpoždění průchodu signálu sítí. Proto je třeba přesný čas časového servera periodicky korigovat pomocí časových informací, získaných z různých časových serverů sítě.

Algoritmy synchronizace hodin vychází z existence přesného časového zdroje. Tímto zdrojem jsou atomové hodiny nebo některý jiný velmi přesný zdroj času. Takovýchto zdrojů je v síti pouze několik. K nim jsou připojeny primární časové servery. Ostatní časové servery získávají časovou informaci zprostředkovaně od sousedních časových serverů. Časová informace se k časovým serverům přenáší též pomocí vysílačů časových značek.

K docílení co možná nejpřesnějšího běhu hodin v časových serverech se používají speciální algoritmy pro synchronizaci.

7.1 Obecná metoda synchronizace hodin

Algoritmus předpokládá, že každý počítač má vlastní časovač C. V době t je hodnota časovače C pro procesor p rovna $C_p(t)$. Ideální stav nastává tehdy, když lokální čas je roven skutečnému času a oba časy mají i shodnou derivaci.

Tedy $t = C_p(t)$ a $\frac{dC_p(t)}{dt} = 1$.

Hodnota času, kterou ukazují lokální hodiny reálného času se však od přesné hodnoty liší. Relativní chyba časovače je dána výrobcem a bývá řádově 10^{-5} až 10^{-6}.

Existuje tedy konstatna ρ taková, že platí

$$1 - \rho \leq dC/dt \leq 1 + \rho$$

kde ρ je maximální rozsah driftu (dáno výrobcem).

Existují-li dva počítače s driftem ρ (opačným), pak po době Δt po synchronizaci se mohou lišit o

$$\delta = 2 * \Delta t * \rho \ [s]$$

Nemají-li se hodiny dvou počítačů lišit o více ne δ [s], pak musí docházet k synchronizaci

ne déle než za

$$\Delta t = \frac{\delta}{2\rho}\ [s]$$

7.2 Cristiansův algoritmus

K synchronizaci dochází periodicky po $\frac{\delta}{2\rho}$ [s]. Hlavní problém seřizování chodu hodit spočívá v tom, že hodiny nesmí jít pozpátku. Proto dojde-li k takovéto situaci, jsou hodiny zpožďovány postupně. Nejčastěji tak, že se odchylka rozpočte rovnoměrně do každého tiku hodin až do další synchronizace.

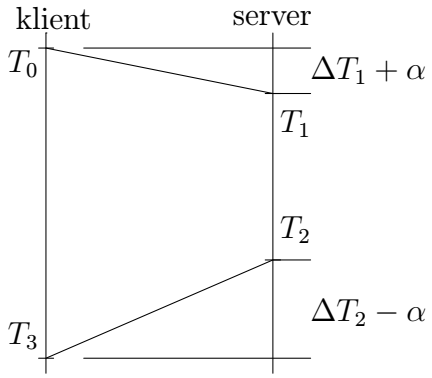
Dalším problémem je odhad doby přenosu a doby obsluhy. Odhad doby odezvy stanovíme podle vztahu

$$\Delta\, T \;=\; (T_1 \;-\; T_0)/2$$

Známe-li dobu obsluhy I, pak můžeme předchozí odhad zlepšit

$$\Delta\, T \;=\; (T_1 \;-\; T_0 \;-\; I)/2$$

K přesnějšímu odhadu doby přenosu provádíme několik měření. Z naměřených hodnot odstraníme ty, které jsou příliš dlouhé. Výslednou hodnotu stanovíme jako aritmetický průměr.

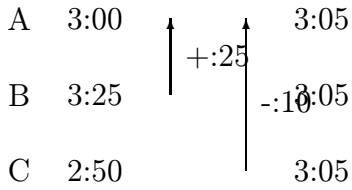


Obrázek 7.1: Stanovení času podle Cristiansova algoritmu

7.3 Pasivní časový server

7.3.1 Berkeley algoritmus

Berkeley algoritmus předpokládá aktivní časový server, který posílá periodické dotazy na čas klientů. Na základě dotazů počítá průměrný čas a všem počítačům posítá zprávu o kolik si mají posunout lokální čas. Používá se v případě, že neexistuje přesný zdroj času. Slouží pouze ke srovnání času na spolupracujících serverech.



Obrázek 7.2:

7.3.2 Algoritmus průměrování

Algoritmus průměrování patří mezi decentralizované algoritmy. Čas je rozdělen na fixní resynchronizační intervaly. Je-li R systémová konstanta, pak pro čas t, ležící v i-tém intervalu platí

$$T_0 + iR \leq t \leq T_0 + (i+1)R$$

V tomto intervalu každý počítač vysílá svůj čas jako zprávu se všeobecnou adresou. Každý počítač po přijetí všech zpráv daného intervalu vypočte průměrný čas.

Zpřesnění vypočteného času se dá dosáhnout eliminací hodnot, které se příliš od ostatních liší (omezení rozptylu). V tomto případě se vypouští se m nejnižších a m nejvyšších hodnot.

Dalšího zpřesnění lze dosáhnout tak, že se přihlíží k době zpoždění přenosu zprávy. Zpoždění se dá odhadnout z topologie sítě nebo podle doby odezvy echa.

Kapitola 8

Distribuované algoritmy

Algoritmy můžeme rozdělit podle prostředí, ve kterém jsou vykonávány na

- paralelní algoritmy, prováděné v multiprogramním prostředí, kdy jednotlivé procesy komunikují prostřednictvím společné paměti
- distribuované algoritmy, prováděné ve víceprocesorovém prostředí, kdy jednotlivé procesy komunikují prostřednictvím zpráv.

Realizace distribuovaných algoritmů způsobuje z hlediska operačních systémů potíže v následujících oblastech

- při řešení algoritmů vzájemného vyloučení
- při řešení problémů se zablokováním (uvíznutím) procesů
- při ukončování procesů
- při manipulaci s kopiemi dat
- při zajišťování konzistence distribuovaného rozhodování
- při řízení přenosu dat.

8.1 Charakteristika distribuovaných systémů

- neexistuje globální stav distribuovaného systému. Neexistuje totiž společná paměť, která by dovolila tento stav zachytit.
- procesy se řídí vnitřními událostmi a událostmi spojenými s přijetím nebo vysláním zprávy.

Distribuovanost je buď vrozena danému problému, nebo je dána způsobem realizace a systémem, na němž je algoritmus realizován.

Rozlišujeme

- distribuovanost dat
- distribuovanost řízení (neexistuje žádný hlavní, proces)

Distribuovanost dat lze chápat jako

- duplicitu dat
- rozložení dat v jednotlivých místech systému

8.2 Základní prvky distribuovaných alogoritmů

V distribuovaném prostředí jsou spolu svázány dvě komponenty

- procesy - sekvenční procesy, které reagují na určité události
- komunikační spoje - které zajišťují informační vazby mezi mezi procesy

Vlastnosti spojů lze chápat staticky (topologie), nebo dynamicky (chování při přenosu dat).

- statické vlastnosti spojů (topologie)
  - úplné polygonální (úplné vzájemné propojení)
  - neúplné polygonální (neúplné vzájemné propojení)
  - stromové (mezi dvěma uzly existuje právě jedna cesta)
  - hvězdicové (jeden hlavní - centrální uzel)
  - pravidelné (např. pravoúhlé, kubické, ...)
  - kruhové (jednosměrné, obousměrné)
- dynamické vlastnosti spojů

  chybovost spojů
  ochrana proti přenosu duplicitních zpráv
  zabezpečení sekvenčnosti přenosu
  zabezpečení proti chybám
  časové parametry při přenosu zpráv
  ohraničenost zpoždění
  ochrana proti zahlcení
  ochrana proti zablokování

Nezabezpečuje-li komunikační programové vybavení odpovídající vlastnosti spojů, je nutno tuto problematiku zahrnout do další programové vrstvy.

8.3 Vlastnosti distribuovaných algoritmů

Mezi vlastnosti distribuovaných algoritmů počítáme

- stupeň rozdělení
- odolnost proti poruchám
- požadavky kladené na síť

8.3.1 Stupeň rozdělení

Rozlišujeme

symetrické uzly - uzly se od sebe nedají rozpoznat jak z hlediska vykonávaných funkcí, tak i z hlediska požadovaných funkcí.

asymetrické uzly - uzly jsou něčím vyjímečné, proto je třeba při požadavku využití této vyjímečnosti určit přímo požadovaný uzel. Příkladem je model server - klient.

Symetrii lze dále jemněji rozlišit

textová symetrie - programy se neliší, označování objektů různými jmény

silná symetrie - stejné programy, stejné procesy, chování dle přijatých zpráv

úplná symetrie - stejné programy, procesy se chovají úplně stejně.

8.3.2 Odolnost proti poruchám

Odolnost proti poruchám souvisí se symetrií a asymetrií - symetrie podporuje zastupitelnost jednotlivých komponent

8.3.3 Požadavky kladené na síť

- provoz v síti (počet zpráv/s, hustota provozu, doba čekání na odpověď)
- dokonalost a nedokonalost prostředí
- globální a lokální stavy (schopnost procesu vytvářet rozhodnutí bez znalosti globálního stavu - snižování počtu zpráv)

8.4 Přístupy a techniky

Přístupy:

- invarianty (kontrola správnosti)
- empirický přístup

Použité techniky zpracování Předpokládá se vysílání se všeobecnou adresou a potvrzování příjmu

8.4.1 Difuzní zpracování

- propojení procesů libovolným způsobem, jeden proces má výlučné postavení - právě on může vyslat první zprávu

- ostatní mohou vyslat zprávu pouze tehdy, když ji přijaly.

- výlučný uzel je kořen grafu - vysílání všesměrové zprávy ostatním uzlům

- používá se pro algoritmy detekce ukončení procesu a vzájemného vyloučení dvou nebo více procesů.

Algoritmus

- kořenový proces pošle zprávu všem svým následníkům dle topologie

- po přijetí pošle každý uzel zprávu všem svým následníkům a čeká na odpověď. Po přijetí odpovědi ode všech svých následníků pošle svou odpověď svému předchůdci.

- je-li proces list stromu, odpovídá okamžitě.

- zpracování končí v okamžiku, kdy kořen získá odpověď ode všech svých následníků.

8.4.2 Obíhající příznak (token, pověření)

- příznak představuje výlučné právo nebo prioritu

- procesy jsou uspořádány do kruhu (fyzického nebo logického). V pořadí tohoto uspořádání je předáváno pověření.

- kruh může být statický, nebo dynamický

- používá se pro algoritmy detekce ukončení nebo vzájemného vyloučení.

8.4.3 Časová razítka

- uzly mohou být libovolně uspořádány

- časová razítka generovaná jedním procesem tvoří monotóně rostoucí posloupnost hodnot. Jejich generování musí být organizováno tak, že se nikdy nesmí vyskytnout dvě stejná.

- používá se pro algoritmy vzájemného vyloučení a detekce uvíznutí.

Předpoklady

- každý proces vlastní logické hodiny (P_i, h_i). Každá zpráva je označena trojicí (m, h_i, i), kde m je zpráva, h_i je časové razítko a i je číslo procesu.

  Řízení hodin - pro řízení hodin platí následující pravidla

- přijme-li proces zprávu (m, h_j, j), nastaví dobu příchodu zprávy na

$$h_i = max(h_i, h_j) + 1$$

- posílá-li P_i zprávu (m, h_i, i), zvýší hodnotu h_i o 1

- hodnota hodin může být zvýšena mezi dvěma vnitřními událostmi kdykoliv, nemá to však význam.

Tímto způsobem je dosaženo částečného uspořádání událostí. Úplné uspořádání dostaneme doplněním časového razítka o prioritu jednotlivých uzlů (mohou to být např. adresy uzlů).

8.5 Synchronizace procesů

Rozeznáváme synchronizační operace (např. semafory) a synchronizační nástroje (např. přenos zpráv).

Existují dvě základní synchronizační úlohy

- vzájemné vyloučení procesů

- volba jednoho ze skupiny procesů

8.5.1 Vzájemné vyloučení procesů, kritické sekce

Algoritmus Ricarda a Agrewaly Předpoklad:

- úplně propojená síť,

- bezpečný přenos dat

- proměnné zpoždění zprávy

- pořadí zpráv není třeba dodržovat

Algoritmus: Řešení vychází z metody předávání pověření. Na počátku je pověření přiděleno náhodně. Každý proces, který chce aby mu byl přidělen příznak posílá ostatním procesům požadavek s časovým razítkem. Každý proces si udržuje pole s informací, kdy ostatní procesy naposledy žádaly o přidělení pověření. Příznak je reprezentován polem časových razítek, kdy byl příznak kterému procesu naposledy přidělen. Opouští-li proces P_i kritickou sekci, hledá od i (cyklicky) první hodnotu, pro kterou platí, že čas posledního požadavku P_l pro vstup do kritické sekce je větší, než hodnota l-té položky v příznaku. Pak předá příznak procesu l.

Příznak je jen jeden, všechny procesy znají svoji identitu, jedná se o textovou symetrii.

Ztráta pověření vede k uváznutí. Detekce ztráty se provádí jedním z následujících postupů:

- doba držení pověření je časově omezena - po uplynutí této doby je třeba spustit algoritmus obnovy pověření.

- algoritmus dvou příznaků s rozdílnou dobou periody. Privilegium má jen jeden. Během určité doby se musí příznaky setkat. Při ztrátě opět nastává fáze obnovy.

8.5.2 Volba jednoho ze skupiny procesů

Problém lze řešit

- centralizované metody - (jeden význačný proces umístěný v centrálním uzlu) - jednoduché z hlediska realizace, problémy při výpadku uzlu.

- decentralizované metody - nutnost naléze centrální uzel - volební algoritmy (na základě jednoznačného identifikátoru procesu)

Changův a Robertsův algoritmus

- předpoklady - jednosměrný kruh, jednoznačné označení všech uzlů.

Algoritmus

- libovolný z procesů (může jich být i více) zahájí výběr vysláním zprávy zahaj_volbu (claim token), ve které vyšle svůj jednoznačný identifikátor.

- přijme-li uzel zprávu zahaj_volbu, porovná svůj identifikátor s identifikátorem zprávy a odešle zprávu s větším z nich.

- proces, který přijme zprávu s vlastním identifikátorem je zvolený proces.

- minimální počet vyslaných zpráv je $o(n)$ průměrný počet vyslaných zpráv je $o(nlgn)$ maximální počet vyslaných zpráv je $o(n^2)$

Hirschbergův a Sinclairův algoritmus

- předpoklady - obousměrný kruh, jednoznačné označení všech uzlů.

Algoritmus

- libovolný proces vyšle zprávu zahaj_volbu do obou směrů.

- proces v větším číslem vyšle do obou směrů svou zprávu zahaj_volbu

- proces s nižším číslem zprávu pouze předá dál

Algoritmus Doleva, Klawa a Rodeha

- předpoklady - jednosměrný kruh, simuluje obousměrný kruh

  Jestliže platí $P_k = max(P_i, P_k, P_l)$, pak P_i ošle číslo P_i, jinak zůstane pasivní.

8.6 Uváznutí, detekce a zotavení

Principy uváznutí

8.6.1 Apriorní metody

1. Lometovy algoritmy
   Apriorní metody
   1. Algoritmus Rosenkranze, Stearnse a Lewise

Algoritmus pracuje na základě porovnání časových razítek, přidělených v době vzniku procesů. Proces, který žádá o přidělený prostředek, může být zrušen, nebo postaven do fronty. Proces, který prostředek vlastní, může být zrušen, nebo je mu dovoleno svoji činnost ukončit a prostředek uvolnit. Podle toho existují dvě metody, jak předejít uvíznutí procesů.

Předpokládejme, že si proces P_1 přidělil nějaký prostředek a proces P_2 o něj právě žádá.

a) wait-die. Je-li proces P_2 starší než P_1, pak P_2 čeká dokud P_1 prostředek neuvolní. Je-li P_2 mladší než P_1, je P_2 zrušen a musí se provézt znovu se stejným časovým razítkem.

b) wound-wait. Pokud je P_2 starší než P_1, pak je P_1 zrušen. Je-li P_2 mladší než P_1, pak P_2 čeká na uvolnění prostředku procesem P_1. Hlavní zásadou, podle které se přidělení žádaného prostředku řídí je, že starší proces nikdy na prostředky držené mladším procesem nečeká.

8.6.2 Aposteriorní metody

Aposteriorní metody vychází z hledání cyklů v grafu závislostí typu prostředek - proces - prostředek - . . .. Pokud je graf lokální, dají se závislosti odhalit. Problém je s globálními uvíznutími, které se projeví vazbou přes několik uzlů.

- Algoritmy s centralizovaným řízením
- Algoritmy s hierarchickým řízením
- Algoritmy s distribuovaným řízením

8.6.3 Uvíznutí při výměně zpráv

V tomto případě se prohledávají závislostní množiny typu dotaz - odpověď.

Podmínky:

- konečné zpoždění zprávu
- spolehlivý přenos
- zachování pořadí zpráv

Při vyhledání uvíznutí se využívá difuzní zpracování. Závislostní množina se detekuje tak, že všechny procesy v ní musí odpovědět, že čekají na zprávu.

Kapitola 9

Metody ochrany zdrojů

Operační systémy vyžadují ochranu ne třech úrovních.

1. **ochrana zdrojů** - ochrana proti neautorizovanému použití prosředků v operačním systému
2. **bezpečná komunikace** - vlastní ochrana přenášené informace
3. **ověřování uživatelů** - zabezpečení, aby zprávy přicházely od ověřeného zdroje a byly přenášeny bez modifikace

9.1 Napadení systému

Napadení systému může být provedeno dvojím způsobem.

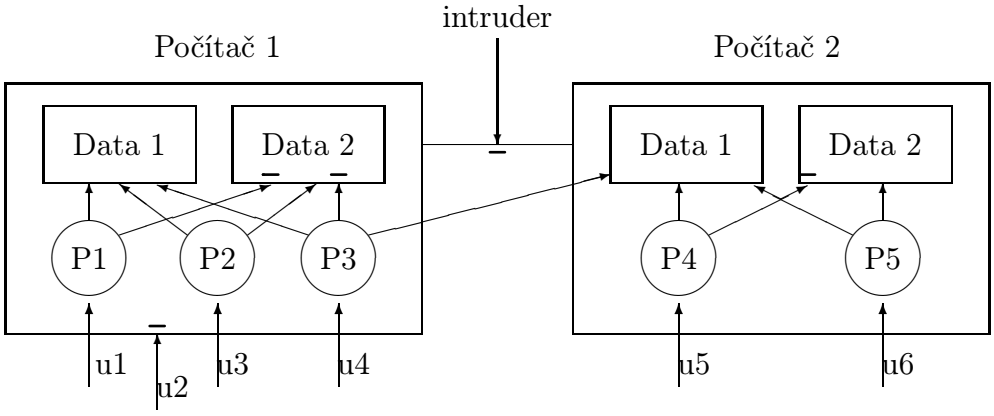
- pasivně (odposlech)
- aktivně (modifikace, podstrkávání zpráv)

Místa napadení schematicky ilustruje následující obrázek.
Krátkou čárkou jsou označena místa napadení uvnitř i vně systému.

9.2 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. V tomto případě je prevence velmi obtížná. Detekovat únik informace je možný např. statistickým sledováním informačních kanálů.
2. **Listování** (Browsing) - intruder se pokouší o čtení informace, uložené v systému. Data nemodifikuje, pouze čte všechny přenášené informace. Opět je obtížné detekovat, ochrana se provádí šifrováním přenášených dat a volbou bezpečných cest.



Obrázek 9.1:

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 uživael nebo program. Cílem je získat přístup k systémovým objektům. Typickým představitelem tohoto způsobu napadení je Trojský kůň.

9.3 Ohodnocení bezpečnosti

K ohodnocení bezpečnosti existuje více způsobů. Zde uvedeme TCB (Trusted Computing Base), popsaný v tzv. Oranžové knize (The Trusted Computer Evaluation Criteria - Orange Book).

Tento systém představuje úplný ochraný mechanizmus ve výpočetních systémech. Zahrnuje programové i technické vybavení a firmware. Je brán v úvahu při návrhu operačních systémů, výrobci jej používají k označení bezpečnosti nabízených systémů.

Systémy jsou podle této klasifikace rozděleny do čtyř skupin. Skupina A je nejlepší, skupina D je bez zabezpečení.

Skupina D Zahrnuje operační systémy bez zajištění bezpečnosti, s minimální ochranou. Typickým představitelem je operační systém MS-DOS.

Skupina C Jedná se o systémy s tzv. volnou ochranou. Stupeň zabezpečení je ponechán na uvážení. Skupina se dále dělí na dvě podskupiny.

Třída C1 Systém s volnou ochranou bezpečnosti. Odděluje pouze uživatele od dat. Chrání data před neautorizovaným přístupem ostatních uživatelů.

Třída C2 Systém s řízenou ochranou přístupu. Uživatel se do systému přihlašuje a jeho pravost se ověřuje pomocí hesla. Další ověřování uživatele se neprovádí. Mechanizmus bezpečnosti je oddělen.

Skupina B Představuje systém s nařízenou nebo vynucenou ochranou. Musí se kontrolovat všechny přístupy ke všem objektům. Mechanizmus ochrany musí být sám o sobě bezpečný. Dále musí být realizována koncepce monitoru odkazů. Skupina je rozdělena na tři podtřídy.

Třída B1 Je systém s tzv. značenou ochranou bezpečnosti. Všem objetům jsou přiděleny klasifikační značky. Přístup subjektů (procesů) k objektům (zdrojům) je řízen. Uživatel má přístup pouze k objektům, které jsou klasifikovány níže, než je on sám.

Třída B2 Je systém se strukturovanou ochranou. Celý systém je od počátku psán podle formálního modelu bezpečnosti. Musí být celý dokumentován. Každý kanál, který může ohrazit bezpečnost systému musí být idnetifikován (např. šířkou pásma). Třída B2 představuje rozumnou úroveň pro zabezpečení.

Třída B3 Je systém, který má vytvořeny oblasti bezpečnosti (Security Domains). Operační systém musí obsahovat tzv. monitor odkazů (reference monitor). Jsou také vytvořeny oblasti bezpečnosti, tj. seznamy uživatelů a skupin s jejich přístupovými právy k objektu, a dále seznam uživatelů a skupin, pro které není zaručen žádný přístup.

Skupina A Je tzv. verifikovaná ochrana. Vyžaduje úplný formální návrh systému, který je orientován na klasifikaci informace. Skupina A zahrnuje jednu třídu.

Třída A1 Je systém s verifikovaným návrhem. Jedná se o obdobu třídy B3 s úplným formálním návrhem, který zaručuje úplnou bezpečnost systému.

V praxi se můžeme setkat i s dalšími způsoby hodnocení bezpečnosti. Některé jsou uvedenému systému velmi podobné.

9.4 Ochrana zdrojů v distribuovaném systému

Ochrana zdrojů v operačním systému se provádí:

- přístupovou maticí
- přístupovým seznamem
- seznamem schoností (kapability)

Přístupová matice obsahuje

- model informačního toku
- objekt, jeho typ a povolené operace
- subjekty, které mají právo manipulovat (procesy, uživatelé)
- přístupová práva a oblasti jejich použití

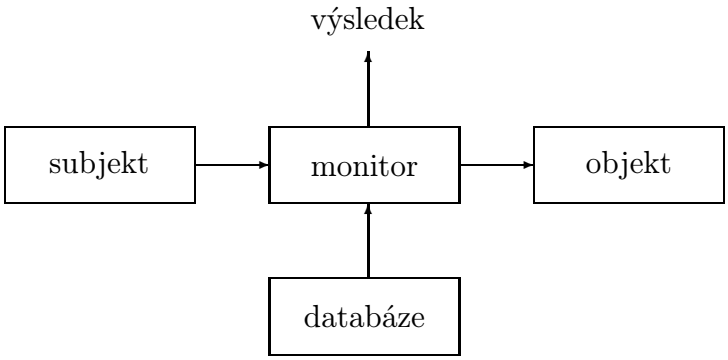
Přístupová matice může být vytvořena staticky, nebo (častěji) dynamicky. V případě dynamického vytváření matice musí být definována práva modifikace matice.

Protože je přístupová matice většinou řídká, nahrazuje se tzv. přístupovým seznmem nebo přístupovým vektorem, který nezahrnuje prázdná pole. Navíc je zaveden subjekt ostatní.

Model informačního toku zahrnuje

- objekty
- procesy
- třídy bezpečnosti
- relace toku

Následující obrázek ukazuje mechanizmus bezpečného jádra.



Obrázek 9.2:

9.5 Způsoby napadení sítě

Intruder může napadnout síť aktivně nebo pasivně.

1. **Pasivní napadení** může být realizováno jako

   a) kradení nebo únik informace. Cílem intrudera je v tomto případě získání obsahu zprávy.

   b) analýza přenosu spočívá v tom, odkud, kolikrát, kam, s jakou délkou bloků, jaké množství dat je přenášeno. Využívá se např. tak, že vyvolám nějakou akci, která aktivuje určité služby sítě (nevím, kde jsou instalovány). Pak pouze sleduji tok dat, nemusím analyzovat obsah zpráv, pouze statistické charakteristiky.

2. **Aktivní napadení** znamená nějaké narušení toku dat. Lze jej rozdělit do čtyř skupin.

   a) Modifikace toku dat může spočívat v

- změně obsahu datového toku, kdy provádím syntézu nových datových paketů s cílem získat např. neoprávněný přístup ke zdroji.
- opakováním, kdy změnou adresy se snažím dosáhnout téhož efektu jako proces, který původní zprávu odeslal.
- změně pořadí, kdy modifikuji přenášená data
- rušením a vkládáním, tj. modifikací přenášené informace. Např. v toku dat zachytím přenášený paket a nahradím jej jiným.

b) blokováním přenosu mezi dvěma entitami

c) zadržováním, tj. zpožděným odesíláním zachycených zpráv

d) vytvářením falešného spojení (maskováním se)

Cíle zabezpečení jsou prevence pasivního útoku a detekce aktivního útoku.

9.6 Zajištění bezpečné komunikace

Při zajišťování bezpečné komunikace se musíme zaměřit na následující komunikační komponenty:

1. **Zaměření na linku** spočívá v zabezpečení na linkové úrovni. Protože tuto vrstvu používají všichni uživatelé, musí být vůči nim transparentní. Není proto možné použít u mnohabodových sítí. Zabezpečení se provádí šifrováním informací spolu s generováním kontinuálního toku dat. Únik informace z jedné linky nenaruší jinou. Tento způsob není vhodný pro otevřené systémy, protože není zaručena bezpečnost v koncových a mezilehlých uzlech, kde se inforace musí dešifrovat (linková úroveň).

2. **Zaměření na koncové uzly** spočívá v tom, že informace je šifrována mezi koncovými uzly (na síťové úrovni). Lze použít jak ve dvoubodových, tak i v mnohabodových sítích. Je nutná standardizace šifrování.

3. **Zabezpečení na úrovni spojení** je realizováno na prezentační nebo aplikační úrovni. Dovoluje volit úroveň zabezpečení podle potřeby jednotlivých aplikací. Snižuje nebezpečí odposlechu.

9.7 Šifrování

K šifrování se používají blokové šifry. Dále lze blokové šifry rozdělit rozdělit na symetrické a nesymetrické.

Symetrické metody šifrování

Symetrické metody šifrování využívají ke kódování i k dekódování jediný klíč. Používá se metoda DES, vyvinutá fy IBM v roce 1977. Šifra byla standardizována NBS (National Bureau of Standards). Zpráva je rozdělena do bloků po 64 bitech. Šifrovací klíč má délku 56 bitů,

zbytek tvoří zabezpečení. Šifrování začíná permutací bitů zprávy, následuje vlastní šifrování v 16 úrovních a nakonec se provede zpětná permutace bitů.

Teoretické výpočty ukazují, že 56 bitový klíč je napadnutelný a 128 bitový virtuálně nenapadnutelný.

Při vytváření šifrovacího algoritmu se musí obecně zajistit následující vlastnosti:

- zrušit statistické charakteristiky textu, tj. četnost výskytu, závislost text - šifra.

- musí odolat odvození klíče z pozorování šifrovaného a nešifrovaného textu.

Nesymetrické systémy šifrování

Nesymetrické metody šifrování využívají dva klíče. Jeden slouží k šifrování, druhý k dešifrování. Pomocí šifrovacího klíče není možné zprávu dešifrovat. Proto může být jeden klíč veřejný a druhý tajný. K distribuci veřejného klíče není třeba žádný speciální mechanizmus, předává se běžnou zprávou. Musí se však zavézt protokol ověřování uživatele, aby nemohl veřejný klíč použít někdo nepovolaný.

K šifrování se používá bloková šifra RSA (nazvaná podle jejích autorů Rivesta - Shamira - Adlemana). Používá bloky dlouhé 256 až 650 bitů, přičemž délka při délce šifrovacího klíče 320 až 380 bitů je její zabezpečovací schopnost srovnatelná s kódem DES s délkou klíče 56 bitů.

Teorie šifry vychází z algebry velkých prvočísel.

Výpočet P, N, S musí být jednoduchý, nebo se vytvoří jejich zásoba v podobě tabulky. K jednomu páru (S,N) může existovat několik klíčů P.

9.8 Ověřování pravosti uživatele

Ověřování pravosti uživatele je nutné použít zejména v otevřených sítích, kde nemusí být problém vydávat se za někoho jiného. Aby bylo ověřování spolehlivé většinou se provádí pomocí speciálních uzlů se zvýšenou bezpečností.

Aktivity vetřelce se mohou projevovat různým způsobem.

1. vetřelec si zapamatuje přenášenou informaci a později ji použije

2. vetřelec blokuje tok informací

3. vetřelec modifikuje tok informací, např. změnou adresy nebo dat

4. vetřelec syntetizuje zprávu a skrývá se za legálního uživatele.

Příjemce zprávy musí být schopen:

- ověřit pravost zprávy, většinou pravost odesílatele

- zjistit změnu, která nastala během přenosu informace

Proto musí příjemce zprávy používat protokol s potvrzováním zpráv (s potvrzováním potvrzení). K tomuto účelu slouží tzv. ověřovací schemata.

Ověřovací schemata musí obsahovat alespoň jedno tajemství a musí být schopna rozpoznat správně použití tohoto tajemství párovou entitou.

Ověřovací metody

Podle stupně ochrany dělíme ověřovací metody jednoduché a přísné metody. Jednoduché metody jsou založeny na heslech. Přísné metody jsou založeny na šifrovacích metodách.

Přísné metody můžeme dále dělit na

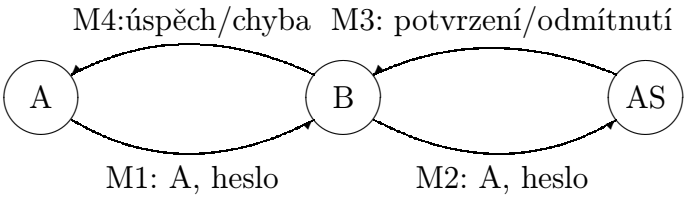
a) elementární metody, které používají symetrických i nesymetrických kódů (veřejných klíčů)

b) metody založené na ověřovacích serverech

c) metody založené na protokolech s minimální znalostí.

9.8.1 Jednoduché ověřování

Jednoduché ověřování je založeno na identifikaci jména a hesla. Přenáší se otevřený text a používá se ověřovací server.



Obrázek 9.3:

Tento jednoduchý způsob ověřování může být předmětem úspěšného pasivního i aktivního napadení. Získané heslo může být použito později.

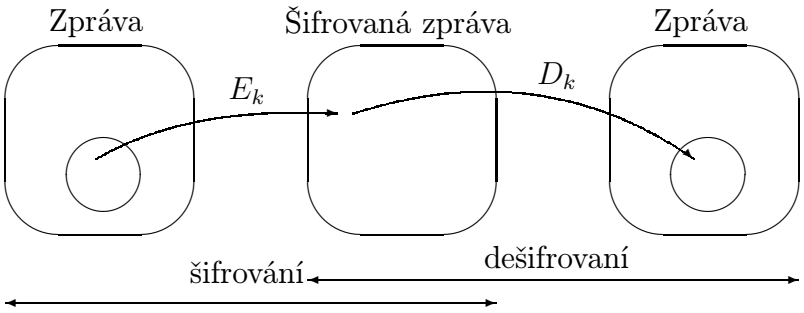
9.8.2 Elementární metody ověřování

Tyto metody jsou založeny na symetrických nebo nesymetrických klíčích.

Ověřování založené na symetrických širovacích 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. Naadne-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.



Obrázek 9.4:

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.

9.8.3 Ověřovací servery

Ověřovací servery slouží k ověřování pravosti uživatele. Přispívají k lepšímu utajení klíčů. Klíče (hesla) se přenáší sítí v šifrované podobě. Funkci ověřovacího servera má počítač, u kterého je zaručena zvýšená bezpečnost. Ověřovací server má instalován modul KDC (Key Distribution Center), ktrý obsahuje databázi klíčů.

Ověřovací server je server, který může doručit identifikační informaci vypočtenou z uživatelského tajného klíče. Hlavní databáze klíčů je indexována podle jmen uživatelů. Obsah databáze je tajný.

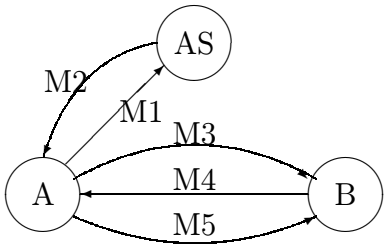
Využití symetrických klíčů k ověřování

Jedná se o obdobu systému pro distribuci klíčů. Každá entita (uzel, proces, uživatel) má přidělen tajný klíč, který zná pouze entita a ověřovací server.

Má-li být ověřeno sopjení mezi dvěma entitami, využívá se znalostí ověřovacího servera, ve kterém musí být obě entity registrovány spolu se svými tajnými klíči. Pro ověřování musí být splněny následující podmínky:

1. Požadavek navázání spojení musí být srozumitelný pouze pro druhou entitu. Pouze druhá etnita může přenesenou zprávu použít pro identifikaci první entity.

2. Druhá entita musí být schopna rozeznat, že požadavek navázání spojení pochází od první entity.



Obrázek 9.5:

Postup ověřování je následující:

1. A posílá AS zprávu M1:{A, B, TI_a} v podobě otevřeného textu.
   A je identifikace (jméno) A
   B je identifikace (jméno) B
   TI_a je identifikátor transakce (zprávy jsou párové)

2. AS posílá A zprávu M2:$\{TI_a, B, K_{ab}, \{K_{ab}, A\}_{K_b}\}_{K_a}$
   K_{ab} je klíč přidělený pro komunikaci mezi A a B
   K_a je tajný klíč entity A
   K_b je tajný klíč entity B

3. A posílá B zprávu M3:$\{K_{ab}, A\}_{K_b}$

4. B posílá A zprávu M4:$\{TI_b\}_{K_{ab}}$

5. A posílá B zprávu M5:$\{f(TI_b)\}_{K_{ab}}$
   $f(TI_b)$ je domluvená funkce pro modifikaci identifikátoru transakce TI_b.

Zprávy M4 a M5 se posílají z důvodu ověření pravosti entity A entitou B. Jinak by mohl zprávu M3 zachytit intruder a použít ji později.

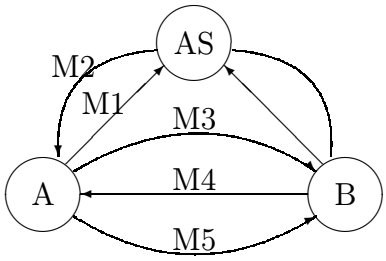
Využití asymetrických klíčů

1. A posílá AS zprávu M1:{A, B}

2. AS posílá A zprávu M2:$\{PK_b,B\}_{SK_s}$

3. A posílá B zprávu M3:$\{TI_a, A\}_{PK_b}$

4. B posílá AS zprávu M4:{B, A}



Obrázek 9.6:

5. AS posílá B zprávu M5:$\{\mathrm{PK}_a, \mathrm{A}\}_{SK_s}$
6. B posílá A zprávu M6:$\{\mathrm{TI}_a, \mathrm{TI}_b\}_{PK_a}$
7. A posílá B zprávu M7:$\{\mathrm{TI}_b\}_{PK_b}$

V předcházejícím obrázku jsou A, B entity, PK_a a PK_b veřejné klíče servera A a B, SK_s je tajný klíč ověřovacího servera a TI_a, TI_b jsou identifikátory transcakcí.

Při ověřování se předpokládá, že
entita A zná svůj tajný klíč SK_a a veřejný klíč ověřovacího servera PK_s
entita B zná svůj tajný klíč SK_b a veřejný klíč ověřovacího servera PK_s
ověřovací server zná svůj tajný klíč SK_s a veřejné klíče PK_a a PK_b.

Šifrování zpráv M2 a M5 tajným klíčen servera SK_s je důležité proto, aby intruder nemohl poslat falešnou zprávu. Šifrování zde slouží k ověření ověřovacího servera (protože ten, kdo nezná PK_s, nemůže zprávu sestavit).

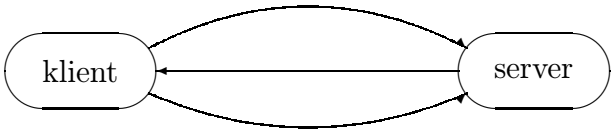
Ověřování založené na protokolech s minimální nebo nulovou znalostí

Základní myšlenkou těchto protokolů je náhrada "znalosti" (hesla, klíče) "znalostí o znalosti". Uživatelé dokazují svoji identitu ne klíči, ale odpovídáním na šifrované otázky servera.

1. klient posílá serveru zprávu M1:{R, ID }
2. klient posílá serveru zprávu M2:$\{\mathrm{C}\}_K$
3. klient posílá serveru zprávu M3:$\{\mathrm{f(C)}\}_K$

R je příznak požadavku
K je tajný klíč
C je nějaké číslo
f(C) je domluvená funkce

Tento systém je použit v ověřování pomocí Kerberosu. Hodnota C se nikdy neopakuje.



Obrázek 9.7:

9.9 Metody rozdělování klíčů

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

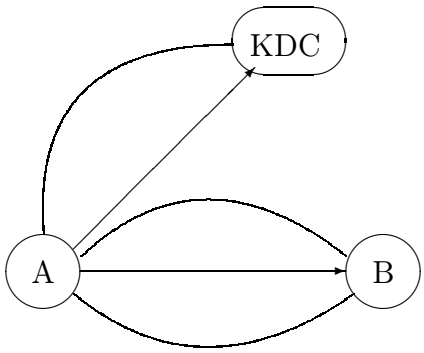
Centrum distribuce klíčů zajistí sdílení klíčů mezi dvěma entitami. Má k dispozici klíč pro komunikaci s každým uživatelem.

Existují tři formy distribuce klíčů:

1. centralizované
2. plně decentralizované
3. hierarchické

Centralizovaná distribuce klíčů

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

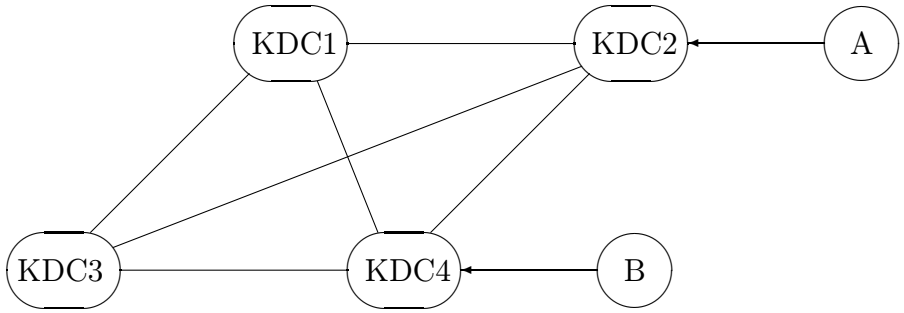


Obrázek 9.8:

A posílá KDC zprávu M1:$\{R_a, ID_a\}$
KDC posílá A zprávu M2:$\{R_a, ID_a, K_{ab}, \{K_{ab}, ID_a\}_{K_b}\}_{K_a}$
A posílá B zprávu M3:$\{ID_a, K_{ab}\}_{K_b}$
B posílá A zprávu M4:$\{RN\}_{K_{ab}}$
A posílá B zprávu M5:$\{TR_t(RN)\}_{K_{ab}}$

Plně decentralizovaná distribuce klíčů

Každý hostitelský systém slouží jako KDC. Vzájemně se informují o přidělování klíčů. Problém je v tom, že máme-li n hostitelských systémů, potřebujeme (n-1)*n/2 klíčů (každý musí komunikovat s každým). Výhodou je, že každý požadavek na přidělení klíče je lokální, komunikace mezi hostitelskými systémy je šifrovaná a spolehlivá. Nevýhodou je příliš mnoho komunikací, potřebných pro distribuci klíčů.



Obrázek 9.9:

Hierarchická distribuce klíčů

V hierarchickém systému přidělování klíčů jsou distribuční centra rozdělena do tří skupin:

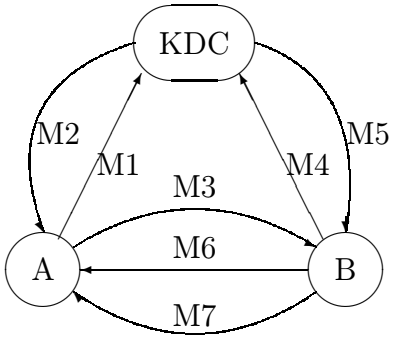
- globální KDC mezi různými oblastmi
- regionální KDC v nějaké oblasti
- lokální KDC pro lokální přidělení.

Distribuce klíčů v asymetrických systémech

I když to vypadá jako zbytečné, i v asymetrických systémech se distribuují klíče pomocí centra distribuce klíčů (KDC - Key Distribution Center). Důvodem je ověřování správnosti distribuovaných klíčů i komunikujících entit. Bezpečnost závisí na správnosti klíče, KDC je oddělen od uživatelů. Klíče je také třeba čas od času měnit, protože při velkém množství přenosů hrozí prozrazení klíče.

Distribuce klíčů pracuje za následujících podmínek:

- každý uživatel zná veřejný klíč KDC
- KDC zná všechny veřejné klíče uživatelů. Soustřeďuje je v tabulce, obsahující páry {uživatel, klíč }.



Obrázek 9.10:

A posílá KDS zprávu M1:{R_a, T}

KDS posílá A zprávu M2:{K_b, R_a, T}$_{KD}$

A posílá B zprávu M3:{A, RS'}$_{K_b}$

B posílá KDS zprávu M4:{R_b, T'}

KDS posílá B zprávu M5:{K_a, R_b, T'}$_{KD}$

B posílá A zprávu M6:{RS, RS'}K_a

A posílá B zprávu M7:{RS'}K_b

kde T, T' jsou časové značky a RS, RS' jsou náhodná čísla.

9.10 Digitální podpisy

Digitální podpisy musí mít následující vlastnosti:

- unikátní
- nepadělatelný

- ověřitelný
- nezapřitelný
- jednoduché metody

Digitální podpisy se realizují dvojím způsobem:

- přímé ověřování podpisu
- nepřímé ověřování podpisu (existuje třetí entita, která řeší spory)

Digitální podpisy jsou založeny jak na symetrických, tak i nesymetrických klíčích.
Digitální podpisy lze použít pro ověřování zpráv a ověřování uživatelů.

9.11 Kerberos

Ověřovací systém Kerberos byl vyvinut v laboratořích MIT v souvislosti s realizací projektu ATHENA. Vzhledem k potřebám zpřístupnit rozlehlý výpočetní systém několika tisícům uživatelů vznikl požadavek spolehlivé a bezpečné identifikace. Kerberos je vlastně protokol pro ověřování uživatelů a zajištění bezpečného přístupu ke sdíleným zdrojům systému.

Při návrhu se vycházelo z předpokladu, že uživatelské systémy je možné napadnout. Pomocí Kerbera musí mít možnost ověřit si důvěryhodnost uživatele, požadujícího službu. Kerberos sám musí být bezpečný, nenapadnutelný výpočetní systém. Sám o sobě neřeší problém napadení, pouze na něj může upozornit.

Síťoví uživatelé vyžadují služby mnoha oddělených počítačů. Z hlediska bezpečnosti existují tři stupně identifikace.

- Povolení přístupu podle počítače. Používá se tam, kde je systém uzavřený nebo tehdy, jestliže organizace kontroluje všechny hostitelské počítače v síti.
- Kontrola identity hostitelským počítačem. Používá se v otevřených systémech. Vychází z předpokladu, že identitu uživatele ověří hostitelský systém, který je pod kontrolou organizace, a který je tím pověřený.
- Kontrola identity uživatele pro každou požadovanou službu. Předpokládá otevřené prostředí, kdy každý hostitelský systém musí mít možnost ověřit klientova práva.

V prostředí Atheny nemusí být hostitelské počítače pod kontrolou organizace. Jednotliví uživatelé mohou mít absolutní práva přistupu ke svému hostitelskému počítači (právo root). Proto se používá kontrola identity pro každou požadovanou službu.

Při realizaci Kerbera byly vytypovány následující požadavky na identifikační mechanizmus.

- Systém musí být **bezpečný**. Kdokoliv, sledující provoz na síti nesmí být schopen získat informaci, která by mu dovolila skrýt se za někoho jiného (maskování).
- Systém musí být **spolehlivý**. Systém služeb jako celek nesmí být napadnutelný.

- Systém musí být **transparentní**. Autorizace uživatele musí být skryta.
- Systém musí být **odstupňovaný**. Nepodporují-li hostitelské programy mechanizmus Athény, nesmí to vézt ke krachu programového vybavení.

Kerberos zajišťuje předchozí požadavky. Počáteční identifikace uživatele je dostatečná. Bezpečnost závisí na bezpečnosti několika serverů, nikoliv na systému, ze kterého se uživatel přihlásí, nebo serverů, které používá.

Autorizační a účtovací schemata mohou být postavena nad ověřováním uživatelů, které provádí Kerberos.

Kerberos vlastní databázi se jmény uživatelů a jejich tajnými klíči. Tajný klíč zná pouze Kerberos a klient, kterému patří. Pro uživatele se klíč vytváří šifrováním hesla. Síťové službym které vyžadují ověřování jsou Kerberem registrovány jako obdobně jako klienti. Tajné klíče jsou dohadovány při registraci.

Vzhledem k tomu, že Kerberos zná tajné klíče, může vytvářet zprávy, které klientům zaručí věrohodnost klienta na druhé straně. Kerberos také generuje dočasné klíče, nazývané *relační klíče*, které jsou určeny pro dva klienty a nikoho jiného. Tyto klíče mohou být použity pro šifrování zpráv.

K zajištění výše uvedených požadavků byly vytvořeny tři úrovně ochrany

1. ověřování při vytvoření spojení (např. spojení se souborovým serverem)
2. ověřování všech zpráv (bezpečné zprávy)
3. ověřování a šifrování zpráv (privátní zprávy, používá sám Kerberos)

Programové komponenty lze rozdělit do následujících celků:

- aplikační knihovny Kerberose
- šifrovací knihovny
- knihovna databáze
- databázové administrační programy
- administrační server
- ověřovací server
- uživatelské programy
- aplikace

K šifrování se používá symetrická šifra podle standardu DES, která je modifikovaná tak, aby nedocházelo k identickému šifrování identických datových bloků. Ze existujících metod se používá metoda CBC (Cypher Block Chaining), při které se nejprve provede nonequivalence nezašifrovaného bloku s předchozím zašifrovaným blokem a následně se takto získaný blok zašifruje.

- CBC - (Cypher Block Chaining) chyba se pozná pouze v daném zašifrovaném bloku.
- PCBC - (Propaged CBC mode) chyba jednoho bloku se přenese do celé zprávy

9.11.1 Databáze Kerbera

Systém řízení databáze je výměnný modul. Původně se používala databáze Ingress.

Kerberos rozeznává dvě databáze. Jednu tajnou, druhou veřejnou. V tajné databázi jsou umístěny informace jako

jméno uživatele

šifrovací klíč, odvozený z hesla uživatele. Má délku 56 bitů plus 8 zabezpečovacích

čas vypršení konta

další administrativní informaci

Veřejná databáze je přístupná prostřednictvím služeb HESOID serveru (jmenný server pro uživatele) a obsahuje další informace o uživateli, jako

celé jméno

telefonní číslo

lokalitu a pod.

9.11.2 Servery Kerbera

Funkce Kerbera jsou rozděleny do několika serverů, které mohou pracovat na několika počítačích. Jsou to

Administrační server se někdy také nazývá *Kerberos Database Management Server*. Zajišťuje čtení a zápis dat do databáze. Správci systému dovoluje přidat uživatele a vypustit jej. Uživateli dovoluje změnit si heslo. Klientská část programu může pracovat na jekémkoliv počítači v síti. Server procuje na stroji, kde je umístěna databáze.

Ověřovací server (Kerberos server), který slouží k ověřování pravosti uživatele a poskytuje mu pověřovací listinu pro přístup k serveru pro ověřování přístupových práv k ostatním serverům systému. Pro účastníky generuje *relační klíče*. Umožňuje databázi pouze číst. Může být umístěn na počítači, kde je uložena kopie databáze určená pouze pro čtení.

HESOID server je rozšířený jmenný server. Navíc obsahuje veřejné informace o uživateli.

Funkce jednotlivých serverů jsou duplikovány. Všechny změny se zapisují do primárního servera, ten pak na pozadí přenáší změny do sekudárních serverů.

Uživatelé mají k dispozici následující programy

kinit přihlášení do Kerbera

kpasswd program pro změnu hesla

klist program pro zobrazování získaných ticketů

kdestroy program pro ničení ticketů

K replikaci databáze se používají programy

kprop - přenos databáze z primarního uzlu do sekundárního

kpropd - příjem databáze v sekundárním uzlu (program je daemon)

Databáze se přenášejí šifrované klíčem, který znají pouze primární a sekundární uzel.

9.11.3 Jména

Jméno uživatele má obecně následující tvar

primarní_jméno.instance@realm

kde

primární jméno - je přihlašovací jméno uživatele nebo jméno služby

instance - je oblast použití jména, např. root, admin, rlogin, ... Instance může být požadovaná služba (rlogin), nebo funkce uživatele (root, admin).

realm - oblast (realm = království). Jméno administrativní entity, která obhospodařuje ověřovací data. Např. athena.mit.edu .

9.11.4 Ověřování Kerberosem

Ověřování Kerberosem probíhá ve třech fázích.

1. získání pověření pro přístup ke službám (credentials - pověřovací listiny)
2. získání pověření pro specifické služby (tickets)
3. předání tiketů příslušnému serveru

Existují dva druhy pověření. První z nich se nazývají *authenticators* - ověřovače totožnosti. Ty získá uživatel od ověřovacího servera při přihlášení se do systému. Druhým typem pověření jsou tickety, které slouží serveru ke kontrole oprávněnosti uživatele využívat požadované služby. Ty získá od TGS (Ticket Granting Server), což je server pro přidělování ticketů.

ticket jednoznačně identifikuje uživatele, kterému byl předán. Používá se namísto jména a hesla. Slouží k ověření uživatele u koncového serveru. V ticketu je též uložena informace, které může být použito k ujištění, že osoba používající ticket je táž osoba, které byl předán. Obsahuje následující položky, šifrované klíčem serveru, pro který je určen.

- S jméno servera
- C jméno klienta
- ($addr_C$) IP adresa klienta

- TS časové razítko
- TTL doba života ticketu
- $K_{S,C}$ náhodný klíč relace

Ticket se může použít i vícekrát, pokud mu nevyprší doba života. Kerberos ticket je tedy n-tice šifrovaná klíčem servera s následujícími prvky

$$\{S, C, addr_C, TS, TTL, K_{S,C}\}_{K_S}$$

Ověřovač obsahuje další informaci, která po porovnání s obsahem tiketu dokazuje, že klient předkládající ticket je týž, kterému byl ticket předán. Obsahuje následující položky, šifrované klíčem dvojice klient, TGS.

- C jméno klienta
- $addr_C$ IP adresu klienta (jeho počítače)
- TS čas vystavení ověřovače (časové razítko)

Ověřovač klienta je tedy n-tice šifrovaná relačním klíčem pro komunikaci klient server s následujícími prvky

$$\{C, addr_C, TS\}_{K_{C,S}}$$

Na rozdíl od ticketu může být ověřovač použit pouze jednou. Chce-li klient využít službu, musí znovu generovat ověřovač. Vzhledem k tomu, že je klient schopen vytvořit ověřovač sám, není v tom problém.

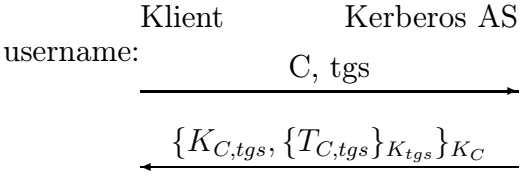
Získání prvotního ticketu Prvotní tiket získá klient od ověřovacího servera na základě identifikace pomocí jména a hesla. Tento prvotní tiket mu pak umožní přístup k dalším službám. Proces ověřování probíhá následovně podle obr. 9.11.

Klient pošle požadavek ověřovacímu serveru (AS). Ten pošle odpověď, šifrovanou klíčem odvozeným z hesla klienta. Klient po obdržení odpovědi převede heslo, zadané uživatelem, na klíč a s jeho pomocí dešifruje odpověď. Dešifrováním získá relační klíč $K_{C,tgs}$ pro komunikaci s TGS a TGT (ticket-granting ticket) $\{T_{C,tgs}\}_{K_{tgs}}$. Obojí si schová a heslo s klíčem K_C vymaže z paměti.

Vyžádání služby Předpokládejme, že klient má ticket pro vybraný server. Aplikace sestaví ověřovač, který obsahuje:

- jméno klienta
- IP adresu klienta
- běžný čas

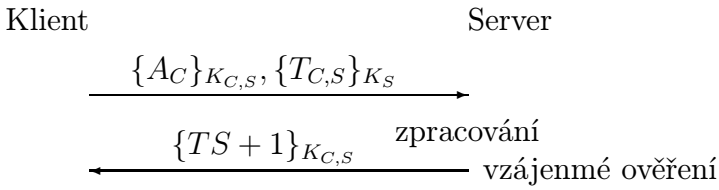
Ověřovač je šifrován relačním klíčem $K_{C,S}$, který obdržel klient od servera pro poskytování ticketů (TGS). Poté pošle ticket spolu s ověřovačem serveru, od kterého klient požaduje službu. Server dešifruje obsah ověřovače i ticketu a porovná je. Ticket může použít klient vícekrát. Ověřovač generuje vždy znovu. Časové razítko v ověřovači slouží k ověření pravosti požadavku.



C	označuje jméno klienta
tgs	označuje požadovanou službu (poskytnout ticket)
$K_{C,tgs}$	relační klíč pro komunikaci klienta s TGS
$\{T_{C,tgs}\}_{K_tgs}$	ticket pro komunikaci klienta a TGS šifrovaný klíčen TGS
K_tgs	označuje klíč servera pro poskytování tiketů (TGS)
K_C	označuje klíč klienta

Obrázek 9.11: Získání prvotního ticketu

Hodiny klienta a servera jsou neustále synchronizovány. Liší-li se příliš aktuální čas od časového razítka, považuje server požadavek za duplikát. Kromě toho může server zaznamenávat historii požadavků podle času a duplicitní požadavky odmítat. Vyžádání služby je schematicky zachyceno na obr. 9.12.



$\{A_C\}_{K_{C,S}}$	ověřovač klienta šifrovaný relačním klíčem pro komunikaci klient server
$\{T_{C,S}\}_{K_S}$	ticket pro komunikaci klienta a servera šifrovaný klíčem servera
$\{(TS+1)\}_{K_{C,S}}$	časové razítko šifrované relačním klíčem pro komunikaci servera s klientem

Obrázek 9.12: Vyžádání služby

Chce-li klient také ověřit pravost servera, přidá server k dešifrovanému časovému razítku z ověřovače jedničku a pošle takto modifikované časové razítko zašifrované relačním klíčem zpět klientovi. Výsledkem je oboustranné ověření pravosti klienta i servera.

Tajný relační klíč je možné použít pro šifrování důvěrných zpráv mezi klientem a serverem.

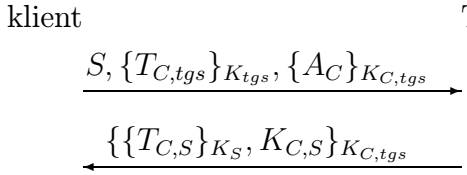
Získání tiketů pro komunikaci se servery Ticket je vhodný pouze pro jeden server. Proto je třeba získat ticket pro každý server, který chce klient použít. Tyto tikety jsou získávány z

TGS (Ticket-granting Server), který distribuuje tikety pro všechny servery. TGS může být realizován jako samostatný server, nebo častěji jako služba na bezpečném serveru (např. na ověřovacím serveru Kerbera).

Má-li klient získat tiket, posílá požadavek na TGS. Požadavek obsahuje

- jméno serveru, pro který je tiket požadován
- TGT (Ticket-granting Ticket), který klient získal od ověřovacího servera, a který potvrzuje klientovu totožnost
- ověřovač sestavovaný klientem

TGS má k dispozici oba klíče pro dešifrování obou částí zprávy. Porovná uvedené údaje a je-li vše v pořádku, vygeneruje nový relační klíč pro komunikaci klienta s požadovaným serverem a nový tiket. Doba života nového tiketu je minimum ze zbývajícího času TGT a implicitní doby přiřazené službě. Schematicky výměnu zpráv ilustruje obr. 9.13.



S	je jméno servera
$\{T_{C,tgs}\}_{K_{tgs}}$	je tiket, získaný od TGS a používaný k ověřování klienta
$\{A_C\}_{K_{C,tgs}}$	ověřovač klienta šifrovaný relačním klíčem pro komunikaci klienta s TGS
$\{T_{C,S}\}_{K_S}$	označuje tiket pro komunikaci klienta s požadovaným serverem
$K_{C,S}$	je relační klíč pro komunikaci klienta se serverem

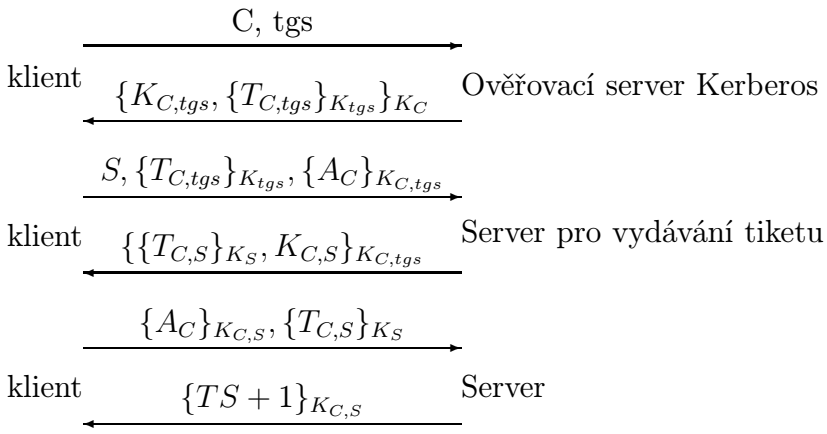
Obrázek 9.13: Získání tiketu pro komunikaci se serverem

Celý proces ověřování při přístupu k serveru ilustruje obr. 9.14.

Administrativní protokol Kerbera Ověřovací servery i servery pro rozdávání tiketů se z důvodu zlepšení spolehlivosti, dostupnosti a průchodnosti replikují (včetně databází). Údržba replik je jednoduchá, protože operace ověřování, které probíhají na primárních i sekundárních serverech, jsou pouze typu čtení. Operace vyžadující zápis do databáze jsou prováděny tzv. administrativní službou - Kerberos Database Management Service (KDBM - Kerberos DataBase Management). Modifikována jsou pouze data primárního KDBM. Neběží-li primární KDBM, nemohou být data modifikována. Požadavky na administační služby přicházejí od uživatelů, kteří si chtějí změnit heslo, nebo od klientské části programu administrátora. Z uživatelského pohledu zajišťují komunikaci programy

- kpasswd - program pro změnu hesla uživatelem. Administrátor může navíc přidávat nebo ubírat uživatele a služby.

- kadmin - klientská část programu administrátora.



Obrázek 9.14: Proces ověřování přístupu k serveru

Server KDBM má následující funkce:

- akceptuje požadavky na změnu účastníků a hesel

- od ostatních serverů se odlišuje tím, že není třeba tiketů

- pro přístup musí být použity ověřovací služby jako získání TGT, vyžaduje jméno a heslo uživatele. Zvýšená ochrana je zavedena proto, aby se nemohl vydávat za administrátora někdo jiný.

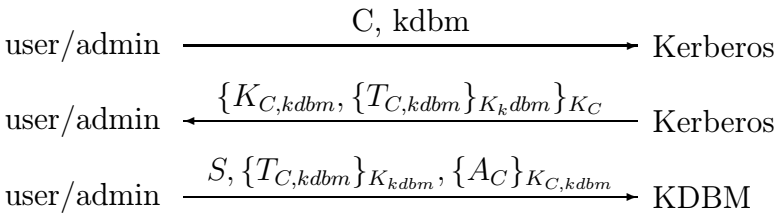
Obdrží-li KDBM požadavek, ověří jméno žádajícího změnu se jménem cíle požadavku. Není-li totožné, zkontroluje je KDBM server s přístupovým seznamem, uloženým v souboru v primárním ověřovacím serveru Kerbera. Jestliže je jméno účastníka nalezeno, je požadavek proveden, jinak odmítnut.

Jména s instancí NULL nejsou podle konvence zahrnuta do přístupového seznamu. Naopak je zahrnuta instance *admin*. Administrátor musí mít vytvořenu instanci *admin*. To dovoluje použít různá hesla pro administraci a pro normální práci.

Všechny přístupy do KDBM jsou logovány.

Administrativní protokol Kerbera ilustruje obrázek 9.15.

Replikace databáze Každá oblast (realm) má primárního servera Kerberos, který udržuje primární kopii databáze. V sekundárních uzlech mohou existovat její kopie. Důvodem k takovémuto uspořádání je lepší dostupnost, rychlejší odezva a zálohování. Duplikací serverů se redukuje možnost vzniku úzkého místa. Konzistentnost kopií databáze je udržována periodickým přepisováním tabulek v sekundárních uzlech. Toto přepisování probíhá posíláním šifrovaných zpráv a opakuje se asi každou hodinu. K přepisu slouží dvě funkce. Ve primárním uzlu to je *kprop*, který posílá obraz databáze. V sekundárním uzlu je to *kpropd*, který je realizován jako daemon.



1. Požadavek KDBM tiketu
2. tiket pro KDBM klienta
3. požadavek klienta na provedení služby kadmin nebo kpasswd

Obrázek 9.15: Administrační protokol Kerbera

Nejdříve posílá *kprop* kontrolní součet nové databáze šifrovaný klíčem, který sdílí primární i sekundární uzly serveru. Pak jsou posílána data. Podřízený vypočte kontrolní součet získaných dat a provede kontrolu.

Kapitola 10

Distribuovaný systém souborů

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, jako jsou čtení, zápis, připojení, ... Adresářové služby manipulují s adresáři. Vytváří a ruší adresáře, vytváří, ruší, mění jména souborů.

10.1 Obecné zásady návrhu distribuovaného systému souborů

10.1.1 Manipulace se soubory

Soubor je možné zobrazit v souborovém systému různým způsobem. Nejrozšířenější je zobrazení v podobě posloupnosti slabik. Tento způsob používají operační systémy jako UNIX nebo MS-DOS. Je nejjednodušší a dovoluje jednoduše manipulovat s daty. Další možností je zobrazit soubor jako sekvenci rekordů. To odpovídá potřebám databázových systémů. Soubor můžeme také zobrazit jako B-strom nebo jako tabulku s přímým přístupem.

Důležitou informací, spojenou se soubory, jsou atributy souboru. Tato informace není součástí souboru, ale obsahuje důležité informace o souboru. Jsou to např. vlastník souboru, velikost souboru, datum vytvoření, přístupová práva a pod. Systém souborů může některé atributy číst nebo do nich zapisovat.

Další oblast problémů v distribuovaných souborových systémech představuje modifikace souborů. V centralizovaných systémech se používá např. technika zámků pro řízení souběžného přístupu. V distribuovaných souborových systémech, využívajících replikaci souborů se řešení problému zjednodušuje zavedením nezměnitelných (immutable) souborů. Zde jsou pak povoleny pouze operace čtení a vytvoření souboru. Operace zápis a připojení jsou zakázány. To vede k zjednodušení manipulace s vyrovnávacími pamětmi a k zjednodušení údržby replikovaných kopií.

Důležitou roli hraje ochrana souborů. Používají se klasické prostředky jako přístupové seznamy (access control list) nebo kapability (capabilities). Přístupové seznamy dělí uživatele

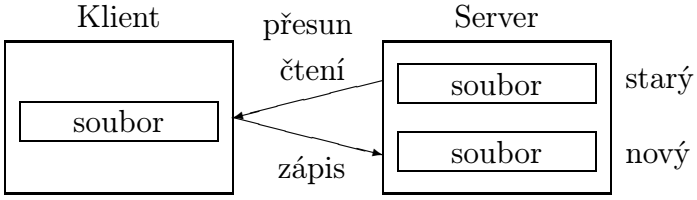
nejčastěji podle vlastníka, skupiny vlastníka, jiný uživatel, jiná skupina superuživatel a ostatní. V některých systémmech může být uživatel členem několika skupin. Přiřazená přístupová práva mohou být čtení, zápis, provedení (execute) a další.

Podle způsobu přístupu k souboru dělíme souborové systémy do dvou skupin.

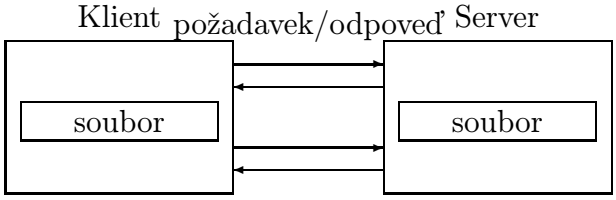
- s přesunem souboru (upload/download)
- se vzdáleným přístupem (remote access)

Systémy s přesunem souboru (upload/download) přesunují celý soubor do vyrovnávací paměti klienta. Při uzavírání souboru přesunuje klient celý soubor na server. Výhodou je jednoduchost systému, nevýhodou časová náročnost při přesunu dat ze servera na klienta (a opačně) a potřeba rezervovat na straně klienta vyrovnávací paměť pro celý soubor. Operace pro manipulaci se souborem se provádí lokálně. Nedovoluje sdílení souborů.

V systému se vzdáleným přístupem jsou přesunovány pouze vyžádané části souboru. Komunikačním systémem jsou přenášeny příkazy pro otevření, uzavření, uzamčení, čtení, zápis, přesun ukazatele i změny atributů. Výhoda je v možnosti sdílet soubory. Nevyžaduje velké vyrovnávací paměti v místě klienta. Nevýhodná je potřeba udržet neustále komunikaci mezi klientem a serverem.



Obrázek 10.1:



Obrázek 10.2:

10.2 Rozhraní pro manipulaci s adresáři

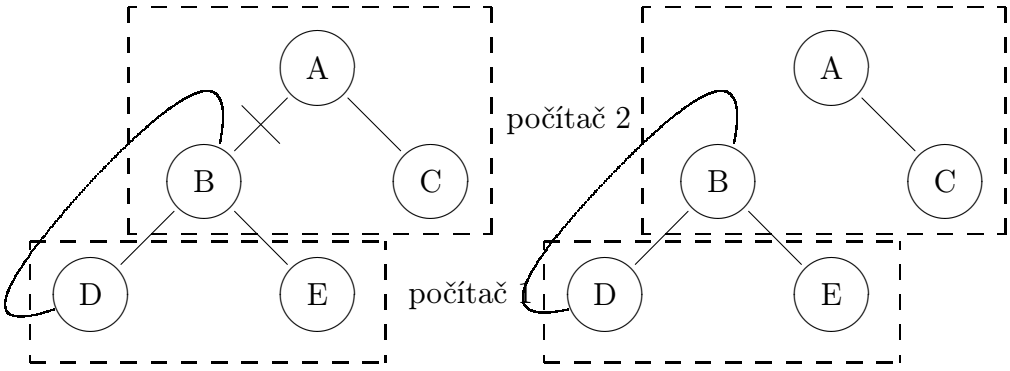
Manipulace s adresáři zahrnuje následující operace

- vytváření a rušení adresářů
- nastavení jména a přejmenování souborů
- přesuny souborů mezi adresáři

Adresářové služby zajišťují převod jména souboru na jeho binární jméno. Pro definici jmen souborů musí být zavedeny nějaké konvence.

Adresářové služby dovolují vytvářet hierarchický systém souborů. V adresářích se nenachází pouze jména souborů, ale i jména podadresářů.

Důležitým prostředkem pro vytváření složitějších struktur adresářů je možnost vytvářet spojení (link). Tato spojení mohou být mezi soubory (jeden výskyt soubor má několik jmen) i mezi adresáři. Spojení mezi adresáři dovoluje vytvářet obecné adresářové struktury (nejen hierarchické). To s sebou však přináší některé problémy. Prvním z nich je možnost výskytu osiřelého stromu (rozpad grafu na dva nesouvislé podgrafy). Může vzniknout např. při odstranění podadresáře, který je spojen (přes další adresáře) sám se sebou. U centralizovaných systémů se tento stav může jednoduše detekovat, protože ke každému adresáři musí existovat cesta od kořene stromu. V decentralizovaných systémech je to problém.

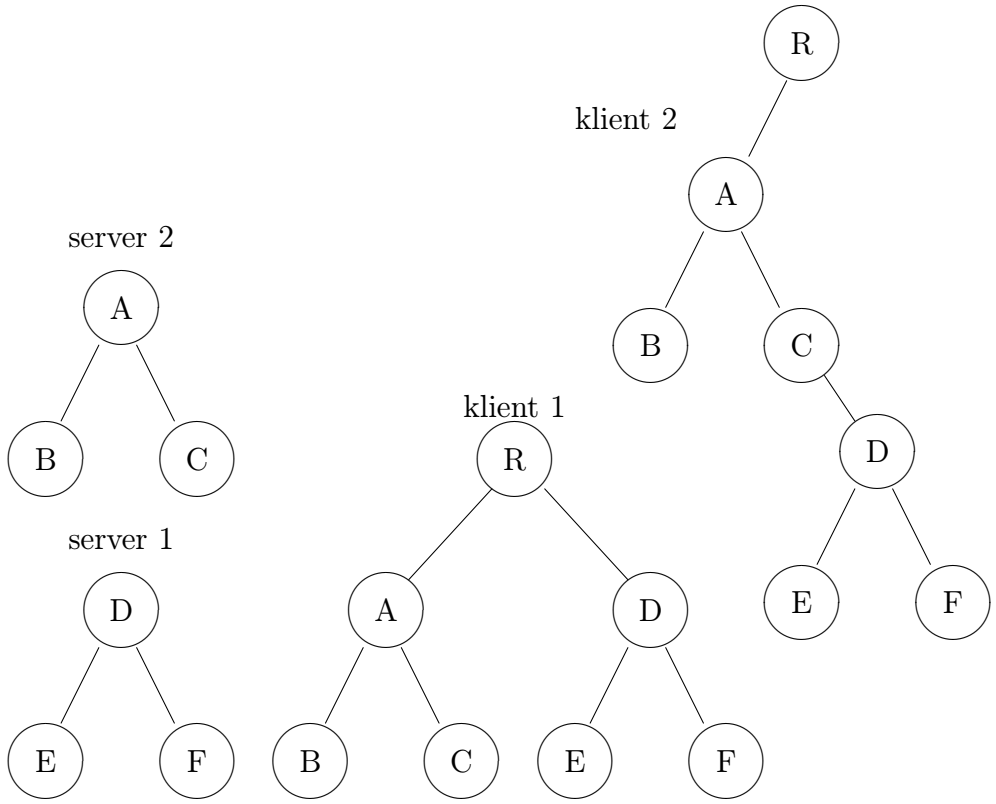


Obrázek 10.3:

Dalším problémem je, jak se jeví struktura adresářů z pohledu jednotlivých klientů. Jsou-li soubory umístěny na více uzlech, je možné je mapovat v jednotlivých klientech různě. To pak může přinášet problémy při zpracování procesů na různých uzlech. Pokud se různému mapování nemůžeme vyhnout, volí se alternativa, kdy podstrom servera se mapuje k podadresáři, který představuje uzel servera (viz AFS).

10.3 Transparentnost jmen

Používáme-li hiearchické adresáře, nejsou jména zcela transparentní vzhledem ke svému umístění na jednotlivých uzlech sítě. Máme dvě možnosti. Jméno souboru je zcela transparentní.



Obrázek 10.4:

Pak jméno servera nesmí být jméno uzlu, na kterém je soubor uložen. Server se může po síti pohybovat, spolu s ním se pohybuje celý svazek souborů. Soubory jsou na jednom serveru, nedají se jednoduše přesunovat. Přesunovat se může celý strom.

Zvolíme-li nezávislé umístění, můžeme přemístit soubory ze serveru na server bez nutnosti měnit jméno cesty.

10.4 Dvouúrovňové označování

V souborových systémech se obecně zavádí dvojí jména. Symbolické jméno slouží k označení souboru "pro uživatele", binární jméno slouží jako identifikátor souboru. Úkolem adresářových služeb je mapování symbolického jména na identifikátor souboru.

V distribuovaných systémech identifikují binární jména server a soubor na tomto serveru. To dovoluje, aby v adresáři jednoho serveru byl umístěn odkaz na soubor, umístěný na jiném serveru (s použitím symbolického linku).

Jako binární jméno je možné použít kapabilit (capabilities). Ty obsahují identifikaci stroje (logické nebo fyzické číslo stroje, reprezentované např. síťoou adresou) a identifikaci souboru (např. ID souboru v UNIXu). Toto uspořádání dovoluje použití jednoho jména pro označení

více souborů. Např. takto můžeme označit originál souboru a jeho zálohy. Použití se nabízí samo - zvýšení odolnosti systému proti chybám redundantními kopiemi souborů.

10.5 Sémantika sdílení souborů

Jestliže jeden soubor sdílí více uživatelů, je třeba definovat sémantiku sdílení souborů, tj. za jakých okolností lze provádět zápis a čtení.

V operačním systému UNIX je přístup k souborům uspořádán podle času, ve kterém byl vyslán požadavek.

V distribuovaných systémech lze dosáhnout obdobného efektu za podmínky existence jednoho servera a bez využití vyrovnávací paměti v místě klienta. Operace se pak provádí jedna za druhou a problémy jsou pouze se zpožděním přenosu v síti. Při praktické realizaci se však server stává úzkým místem.

Předpokládáme-li existenci vyrovnávacích pamětí v místě klienta, můžeme pro sdílení souborů v distribuovaných systémech uvažovat následující sémantiky:

1. sémantika UNIXu
2. relační sémantika
3. sémantika nezměnitelných souborů
4. sémantika transakcí

Relační sémantika Relační sémantika znamená, že změny v souboru jsou viditelné pouze v procesu, který soubor modifikuje. Pro ostatní procesy jsou viditelné teprve po uzavření souboru. Důsledkem je, že z více souběžně provedených oprav je úspěšná ta, která se provedla naposled. Sdílení souborů tak, jak je známe z UNIXu, není v distribuovaném systému možné. Není např. možné sdílet ukazatel na aktuální slabiku v souboru.

Sémantika nezměnitelných souborů V tomto případě pro práci se soubory existují pouze operace vytvoření (create) a čtení (read). Soubory nemohou být modifikovány. Chceme-li provézt změnu v souboru, musíme vytvořit jeho kopii. Problém nasává pouze tehdy, jestiže jeden proces soubor otvírá a druhý jej (jeho novou verzi) zavírá.

Sémantika transakcí Transakce byly rozebrány jako samostatná záležitost. Z pohledu manipukace se soubory se jedná o řešení které nejvíce odpovídá práci se soubory v centralizovaném systému.

10.6 Struktura systému souborů

10.6.1 Moduly

Souborový systém sestává z modulů, realizovaných na serveru a modulů, realizovaných na klientech. Mouhou být umístěny odděleně nebo na jednom uzlu.

Mezi moduly patří

souborový server , který zajišťuje přístup k souborům

adresářový server , který převádí jméno souboru na binární jméno

diskový server , sloužící k vlastnímu uložení souborů a adresářů

caching server , který dovoluje využívat vyrovnávacích pamětí jak při přístupu k datům, tak i při přístupu k adresářům

10.6.2 Způsob prohlížení adresářů

- interaktivní prohlížení adresářů znamená, že klient si vyžádá převod jména na binární jméno, dostane odpověď a opět se obrátí na souborový server s požadavkem přístupu k souboru.

- automatické prohlížení znamená, že se klient obrátí na adresářový server s požadavkem na zajištění přístupu k souboru, adresářový server provede převod jména na binární jméno a sám pošle požadavek na souborový server. Ten zpřístupní soubor přímo klientovi.

10.6.3 Použití vyrovnávací paměti pro převod jmen

Vyrovnávací paměť lze na straně klienta použít i pro ukládání převodní tabulky jméno - binární jméno. Výhoda spočívá v tom, že odpadnou časté dotazy na adresářový server. Nevýhoda je v potenciální možnosti, že převod zastará (např. při zrušení nebo přepsání souboru).

10.6.4 Stavové a bezestavové servery

Souborové servery je možné realizovat jako stavové, nebo bezestavové. Každá z těchto variant má své výhody a nevýhody.

Bezestavové servery:

výhody jsou odolné proti poruchám, mouhou souběžně obsluhovat neomezený počet klientů, jednoduché zotavení po chybě servera i klienta.

nevýhody nutno přenášet stavovou informaci ze servera na klienta a zpět

Stavové servery:

výhody kratší zprávy, vyšší výkonnost, uzamykání souborů

nevýhody omezený počet souběžně obsluhovaných klientů, obtížné obnovení stavu po výpadku servera.

10.6.5 Použití vyrovnávacích pamětí

Vyrovnávací paměti je možné umístit

- na serveru. To má výhodu z hlediska uzamykání částí souborů, při souběžném přístupu k částem souborů, minimalizaci počtu přenosů mezi diskovou pamětí a vyrovnávací pamětí. Nevýhoda je zabudování algoritmu vyhazování stránek (strategie FIFO, LRU apod.)

- na klientu. Výhoda je v možnosti vícenásobného přístupu k datům bez nutnosti je přenášet sítí. Nevýhoda v omezené možnosti uzamčení přístupu k datům a nemožnosti data sdílet.

- někde v síti. Slučuje nevýhody obou předchozích variant. Jediná výhoda by mohla spočívat v omezení počtu přenosů dat sítí.

Vyrovnávací paměti jsou realizovány jako

- polovodičová paměť. Výhoda je rychlý přístup k informaci, nevýhoda ztráta informace při výpadku napájení.

- disková paměť. Výhoda je zachování informace při výpadku (stálá nebo trvalá paměť), nevýhoda v dlouhé době přístupu.

Udržení konsistentnosti vyrovnávací paměti nedělá potíže u servera, protože přístup do ní řídí přímo server. Větší potíže nastávají u klienta. Neexistuje ideální metoda, jak by se měl klient bránit ztrátě informace při výpadku. Situaci lze řešit následujícími postupy

- bezprostředním zápisem do stálé paměti. Nevýhoda je ve značné režii. Každý zápis je podstatně pomalejší než přístup do paměti počítače.

- zpožděným zápisem. Výhoda je v omezení počtu zápisů do stálé paměti. Metoda dovoluje též sdružovat zápisu (blok se zapíše na disk pouze jednou, změn v bloku může být více). Nevýhoda je v nutnosti obnovit stav stálé paměti po výpadku, tj. zkontrolovat konzistentnost celého disku. To může zabrat poměrně dost času (zejména není-li vyrovnávací paměť oddělena od lokálního souborového systému.

- zápisem po uzavření souboru. Tato metoda se používá často. Nedovoluje však sdílení souboru. Její výhoda je v tom, že napočátku se přečte soubor do vyrovnávací paměti, při uzavření se zapíše zpět jako nová verze souboru. Nevýhoda je v tom, že vyžaduje tak velkou vyrovnávací paměť, aby se do ní vešly všechny otevřené soubory.

- centralizované řízení (transakce). Tato metoda byla popsána v jedné z předchozích kapitol.

10.6.6 Zajištění replikace souborů

Pro zavádění relikovaných souborů (mnohonásobné kopie) existují tři důvody:

- zvýšení spolehlivosti vytvářením nezávislých záloh

- zvýšení dostupnosti rozmisťováním kopií do různých uzlů sítě

- odstranění úzkého místa rozprostřením zátěže na více serverů

K dosažení transparentnosti replikací lze použít následující techniky:

- explicitní replikace souborů
- "líná: replikace souborů
- replikace s použitím skupin

Explicitní replikace souborů Programátor má k dispozici tabulku, obsahující informace o názvech a uložení replikovaných souborů. Proces kopírování je celý v jeho režii.

"Líná" replikace souborů Programátor provede modifikaci jednoho souboru, souborový server udělá ostatní kopie. Proces vytváření kopií probíhá na pozadí, bez omezování uživatele. Tento způsob vytváření replik souborů je použit také v technice primární kopie.

Replikace s použitím skupin Z množiny zúčastněných uzlů je vytvořena skupina, která přdstavuje z hlediska provedení zápisu majoritu. Operace probíhá souběžně nad všemi členy skupiny.

Protokol opravy využívá techniku hlasování. Ta vychází z následující myšlenky. Mějme skupinu, jejíž celkový počet hlasů je N (jeden uzel musí mít přidělen 1 nebo více hlasů), N_w je počet počet hlasů, potřebných pro zápis a N_r je počet hlasů, potřebných pro čtení.

Nechť platí

$$N_w + N_r > N \ a \ N_r > N/2$$

První z podmínek zajistí vyloučí souběžný zápis, druhá souběžný zápis a čtení. Volíme-li $N_r = 1$, preferujeme výrazně čtení, které se může uskutečnit z libovolného uzlu. Naopak pro zápis je třeba získat hlasy všech N uzlů. Druhým mezním případem je volba $N_w = N/2+1$, která sice vyžaduje jako souhlas se zápisem minimální počet hlasů, ale ke čtení potřebuje také alespoň $N/2$ hlasů. Navíc jak při čtení, tak i zápisu musí pracovat s nejnovější kopií.

Kapitola 11

Příklady distribuovaných systémů

11.0.7 NFS

NFS (Network File System) lze chápat jako distribuovaný souborový systém. Byl vyvinut firmou SUN Microsystems a dán k dipozici ve zdrojové formě, aby mohl být rozšířen i na ostatní platformy. Poskytuje transparentní vzdálený přístup ke sdíleným souborovým systémům prostřednictví 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 systému, síťové architektuře a transportním protokolu. Tato nezávislost je zajištěna použitím mechanizmu volání vzdálených procedur. Transparentnost přenosu je zajištěna pomocí XDR (eXternal Data Representation).

Doplňkový protokol MOUNT poskytuje funkce specifické pro příslušný operační systém, které umožňují klientu připojit vzdálený strom adresářů do určitého místa v lokálním souborovém systému. Proces mount také umožňuje serveru přidělit přístupová práva omezené množině klientů pomocí protokolu *export*.

Ve vezi 3.0 přibyl modul pro uzamykání souborů NLM (Network Lock Manager), který izoluje stavové aspekty zamykání souborů v odděleném protokolu.

Doplňkem protokolu NFS je NIS (Network Information Service), který zajišťuje jednotnou administraci jmen a hesel uživatelů. Uživatelé se mohou přihlásit k různým počítačům a přitom pracovat se stejnými soubory. Soubory s daty uživatele jsou uloženy na jednom počítači. Administrátor udržuje jejich jednu kopii. Lze vytvořit i více domén, zahrnujících různé skupiny počítačů a uživalelů. Kromě přihlášení se prostřednictvím NIS lze realizovat i lokální přihlášení uživatele na daný počítač.

Data umístěná na jednom počítači mohou být sdílena více uživateli. 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.

Realizace servera vyžaduje pouze podporu komunikačního programového vybavení (RPC, XDR, BSD Sockety). Realizace klienta znamená zásah do jádra operačního systému. Ten musí podporovat VFS (Virtual File System), dovolující mapování vzdálených souborů do lokálního souborového systému.

NFS server pracuje jako bezestavový. Spolu s odpovědí na požadavek klienta vrací deskriptor

souboru (file handle), což je blok dat délky 32 slabik, který jednoznačně identifikuje další požadavek klienta na přístup k souboru. Obsah deskriptoru je závislý na souborovém systému servera. Pro klienta je "nečitelný". Klient jej použije při další komunikaci se serverem. Kromě deskriptoru obsahuje volání klienta jméno požadovaného souboru a identifikaci uživatele (uid, gid), které je použito k určení přístupových práv k souboru nebo adresáři.

K urychlení diskových operací se na uzlu klienta instaluje modul BIOD (Basic Input Output Daemon). Ten provádí asynchronní vstupně/výstupní operace s použitím metody *dopředného čtení* (read ahead) a *opožděného zápisu* (write behind).

Jestliže klient spolupracuje s více servery je běžné, že jeho lokální *uid* a *gid* neodpovídá vzdálenému *gid* a *uid*. Proto je někdy na stanici klienta také spuštěn daemon **ugidd**, který namapuje UID/GID prostor servera na UID/GID prostor klienta. **ugidd** je také založen na volání RPC.

Někdy není vhodné připojit najednou všechny NFS svazky, které by mohl chtít uživatel používat. Důvodem může být omezení maximálního počtu připojených svazků nebo časové důvody. Řešením je použití daemona **automounter**, který v případě potřeby automaticky a transparentně připojí požadovaný NFS svazek a odpojí jej, když není potřeba. Program je schopný též připojit určitý svazek z různých míst. To je vhodné zejména tehdy, jestliže chceme NFS server zálohovat replikami zpřístupňovaných dat umístěnými na různých serverech.

Postup zpřístupnění vzdálených souborů je tedy následující. Nejprve se v počítači klienta vyvolá příkaz *mount*, který ověří uživatelská práva a připojí do adresářového stromu klienta podstrom souborů servera. V dalších fázích komunikace jsou soubory i podadresáře přístupné a je možné přenášet data. Odpojení vzdáleného NFS serveru se provede příkazem *umount*.

Při mapování souborů lze zadat různé parametry, jako např. velikost datagramů pro čtení a zápis, časové omezení pro čekání na vykonání požadavku, hard/soft mapování (má-li klient vyžadovat mapování servera "natvrdo" nebo po vypršení časového omezení podat o neúspěchu pouze hlášení) a pod.

Při spouštění *mountd* jsou parametry nastaveny v souboru /etc/exports. Zde je nutné určit které adresáře, jak a komu budou zpřístupněny. Pro každý takto zpřístupněný adresář je vyhrazen jeden řádek. Zdává se zde jméno nebo adresa počítače, práva (např. rw/ro), způsob ověřování RPC (bez ověření, UNIX, Kerberos), způsob mapování uživatelova uid a gid na uid a gid souborů na serveru a pod.

11.1 Amoeba

Amoeba je distribuovaný operační systém, vyvinutý na Vrije Universiteit v Amsterodamu týmem prracovníků pod vedením prof. A. S. Tanenbauma. Cílem projektu bylo vytvořit systém, který by umožnil využití skupiny heterogenních počítačů jeko jednoho integrovaného výpočetního prostředku. Z hlediska uživatele by se měl takový systém chovat jako jednoprocesorový počítač se sdílením času. Uživatel nemá a nepotřebuje mít žádné vědomosti o tom, kolik a jakých počítačů takovýto systém obsahuje, popř. kde jsou umístěny. Správa a efektivní využití použitých prostředků je zcela záležitostí vytvořeného operačního systému.

Vytvořený výpočetní systém je možné využít jak pro paralelní výpočty, kdy jsou procesory nasazeny na řešení jednoho problému, tak i pro distribuované výpočty, kdy celý systém současně využívá několik uživatelů, pracujících na různých výpočtech, které sdílí společné zdroje.

11.1.1 Architektura systému

Podle funkce lze jednotlivé počítače v systému rozdělit do následujících skupin.

- **procesorový pool** - skupina procesorů s vlastní operační pamětí a s připojením na síť, která tvoří hlavní výpočetní sílu systému. Jednotlivé procesory nejsou vlastněny žádným z uživatelů. Systém pro každý příkaz zadaný uživatelem dynamicky přiděluje jeden nebo více procesorů. Po provedení příkazu se procesory vrací do poolu, odkud mohou být opět přiděleny jinému uživateli. Mezi jednotlivými procesory se nepředpokládá existence společné sdílené paměti.

- **pracovní stanice** - skupina počítačů, sloužících ke komunikaci s uživateli. Pracovním prostředím jsou standardní X-windows, které mohou běžet na obecně využitelných pracovních stanicích, příp. mohou být použity X-terminály.

- **specializované servery** - počítače vyhrazené pro poskytování specializovaných služeb. Jedná se např. o souborové servery, adresářové servery a pod.

- **brány** - slouží k připojení systému AMOEBA k rozlehlým sítím. Je to dáno tím, že Amoeba využívá pro vnitřní komunikaci vlastní komunikační protokol, založený na bázi protokolu Ethernet.

11.1.2 Architektura OS

Amoeba je založena na principu tzv. **mikrojádra**. Jedná se o část operačního systému, která realizuje základní funkce operačního systému. Běží na každém použitém počítači. Ostatní funkce jsou realizovány jako procesy, běžící v uživatelském režimu.

Mikrojádro

Mikrojádro je základní část operačního systému. Běží na všech počítačích. Zajišťuje tyto základní funkce:

- **správa procesů** - každý proces má vlastní adresní prostor. Běh procesu může být rozdělen na několik vláken (threads). Každé vlákno má vlastní sadu registrů, vlastní ukazatel programu a vlastní zásobník. Všechna vlákna běží podle jednoho programového modulu a sdílí společný adresní prostor pro data. Samotné mikrojádro je také realizováno s využitím systému vláken. To dovoluje využívat služeb jádra paralelně (pseudoparalelně). Např. proces, sestávající z několika vláken může v každém vlákně volat jádro operačního systému. Vlákno je "prodlouženo" do jádra. Je-li třeba čekat na ukončení nějaké operace, přeplánuje se na úrovni jádra procesor a spustí se jiné vlákno.

- **komunikace mezi vlákny** Amoeba umožňuje komunikaci mezi dvěma libovolnými vlákny. Vlákna mohou být součástí téhož procesu, různých procesů i různých procesů, běžících na různých počítačích. Ke komunikaci se používají dva základní komunikační modely: RPC, a skupinová komunikace. Komunikace je založena na protokolu FLIP (Fast Local Internet Protocol), který byl k tomuto účelu speciálně navržen.

- **základní správa paměti** - každé vlákno může přidělovat a uvolňovat bloky paměti zvané segmenty. Tyto segmenty mohou být mapovány do adresního prostoru odpovídajího procesu. Přidělené paměťové segmenty jsou rezidentní. Z důvodu rychlosti není podporován žádný způsob virtuální paměti. Maximální velikost prováděných programů je omezena na maximální velikost dostupné paměti.

- **vstupně/výstupní operace** - V/V operace jsou zajišťovány vlákny jádra, které mohou být vyvolány vlákny speciáních procesů. Právo využívat V/V operace mají pouze souborové servery a ostatní systémová vlákna.

11.1.3 Procesy

Procesor je reprezentován deskriptorem procesu. Je to objekt, který má vlastnosti (kapability). S použitím kapabilit může být proces pozastaven (suspended), znovu spuštěn (restarted), zrušen (destroyed) a předán mu signál (signaled).

11.1.4 Objekty

Jednotící koncepcí v systému AMOEBA je koncepce objektů. Jednotlivé servery provádí operace nad objekty.

Objekt je ohraničený kus dat, nad kterým jsou definovány operace. Objekty jsou pasivní, neobsahují ani procesy ani metody. Operace nad objekty se provádí pomocí RPC. Klient specifikuje

- objekt

- požadovanou operaci

- parametry

Operace jsou prováděny synchronně. Klient musí čekat na ukončení operace. Manipulace s objekty je realizována transparentně. Klient nezná umístění objektu, nezná servery, které s objekty manipulují. Informace o objektech jsou uloženy v adresářích (jednotný přístup a další výhody) jako capabilities (kapability, schopnosti).

Na jednom uzlu mohou být spuštěny jak servery, tak i klienti.

Capabilities Objekty jsou označovány a chráněny jednotně - pomocí speciálních tiketů, nazývaných capabilities.

Při vytváření objektu požádá o tuto službu klient servera. Sdělí mu co požaduje a server vytvoří objekt. Jako odpověď vrátí klientovi capabilities. Ty pak v dalších operacích používá klient k označování objektů. Délka pole capabilities je 16 slabik. Jejich strukturu zachycuje následující obrázek.

Server port	Object	Rights	Check
48	24	8	48

Server port určuje požadovaný server. Služba je identifikována číslem. S číslem portu je spojena také informace o uzlu, na kterém server běží. Čísla portů s dalšími informacemi se ukládají do lokální vyrovnávací paměti, aby byly rychle k dispozici. Nenalezne-li se číslo portu ve vyr. paměti, použije se pro vyhledání servera zpráva se všeobecnou adresou. Port může být spojen i s více servery. Pak se vybere pouze jeden z nich. Existují též všeobecně známé porty, se kterými jsou spojeny běžné služby (např. souborové servery). Číslo portu spojené s novou službou je generováno při spuštění servera.

Objekt představuje označení objektu na serveru. Je obdobou i-node v UNIXu.

Rights je bitová mapa povolených operací. Každý typ objetu může využívat jednotlivé bity jinak.

Check je pole pro ověření správnosti capability.

Ochrana objektů Při vytvoření objektu se vytvoří také jeho capabilities. Obsah pole Check se určí náhodně. Práva se nastaví na 1 ... 1 (práva vlastníka, který může vše). Pole Check se uschová v lokální tabulce servera pro budoucí kontrolu. Capabilita se vrátí klientovi.

Klient může vytvořit omezenou kapabilitu pro dalšího klienta. Pošle kapabilitu spolu s novou maskou práv serveru. Server modifikuje práva přístupu k objektu (operací práva XOR maska) a vygeneruje nové zabezpečení kapability s použitím původního zabezpečení a nových práv. Použitá funkce je "jednosměrná", je obtížné najít funkci k ní inversní, která by umožnila uživateli najít původní hodnotu pole Check a modifkovat si tak práva podle vlastního uvážení. Server vytvoří nové kapability a pošle je zpět klientu. Ten je může poslat jinému klientovi, a tak mu zpřístupnit příslušný objekt.

Standardní operace Nejčastěji používané operace nad objekty jsou:

Age - provádí cyklus čištění (odstranění objektů, které již nejsou potřeba nebo přístupné, protože se k nim ztratily kapability)

Copy - vytváří duplikát objektu a vrací nové kapability. Kopie není přenášena přes klienta, ale přímo ze servera na servera.

Destroy - ruší daný objekt a uvolňuje paměť

Getparams - požadavek na poslání parametrů, spojených se serverem

Info - ASCII řetězec popisu objektu

Restrict - poslání nové omezené kapability objektu

Setparams - nastavení parametrů, spojených se serverem

Status - požadavek na zaslání běžného stavu serveru

Touch - dává najevo, že si činí nárok na existující objekt. Používá se při prodlužování doby života objektu (typicky u souborů).

11.1.5 Řízení paměti

Paměť je rozdělena na segmenty. Proces může mít přidělen libovolný počet paměťových segmentů. Segmenty nejsou swapovány ani stránkovány - leží rezidentně v paměti. Každý segmnet je v paměti uložen kontinuálně.

Nad segmenty (jsou chápány jako objekty) jsou definovány operace pro:

vytváření - vytvoření segmnetu, vrací kapability segmentu. Ty se pak používají ve všech zbývajících operacích.

rušení - uvolnění segmentu

čtení - čtení bloku dat ze segmentu

zápis - zápis bloku dat do segmentu (zavedení programového modulu)

Při startu má proces jeden segment, za běhu si může vytvářet další a mapovat je do svého logického adresního prostoru (**mapování**, **odmapování**, **přemapování**). Segmnet může být mapován do logického prostoru více procesů. To dovoluje **sdílení** paměti. Běžně se ale používá sdílení paměti mezi vlákny jednoho procesu. Při sdílení paměti mezi více procesy je třeba zajistit také ochranu přístupu.

11.1.6 Komunikace

Mikrojádro zajišťuje komunikaci mezi libovolnými dvěma vlákny patřícími libovolným procesům v systému. Základní komunikační mechanizmus je RPC.

Volání vzdálených podprogramů Komunikační model RPC slouží k synchronní komunikaci mezi dvěma vlákny. Vlákno zahajující komunikaci se nazývá **klient**, volané vlákno **server**. Klient v podstatě vyžaduje vykonání nějaké operace serverem, tento požadavek provádí zasláním zprávy serveru. Server provede požadovanou operaci a odešle klientovi zprávu o výsledku. Klient je v době mezi odesláním požadavku a přijetím odpovědi zablokován.

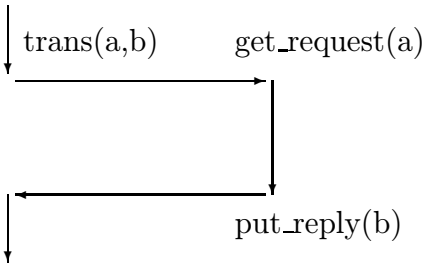
Pro komunikaci pomocí RPC Amoeba poskytuje speciální jazyk AIL (Amoeba Interface Language) - klient vyvolá knihovní proceduru, spojku. Spojka převezme parametry a vytvoří zprávu, kterou předá jádru OS na daném počítači. Jádro odešle zprávu na počítač, kde běží proces server. Zde je zpráva přijata místní kopií jádra, které ji předá spojce servera. Server provede požadovanou operaci s dodanými parametry, výsledek umístí do zprávy a stejnou cestou ji pošle klientovi.

Při vytváření vlákna voláním jádra si vlákno vybírá logickou adresu, tzv. **port**. Porty všech vláken běžících na počítači jsou registrovány v jádře na tomto počítači. Zprávy posílané pomocí RPC jsou adresovány z portu na port, jádro zjišťuje fyzickou síťovou adresu volaného portu pomocí broadcast zprávy **locate**. Na tuto zprávu odpoví jádro počítače, kde je hledaný port registrován svojí síťovou adresou. Volající jádro informaci uloží do své směrovací tabulky. Při dalším volání se již **locate** neposílá.

Pro komunikaci pomocí RPC jsou zavedeny následující primitivy:

- **get_request** - server čeká na příkaz

- **port_reply** - odeslání odpovědi serverem
- **trans** - zpráva od klienta k serveru - čeká na odpověď.



Obrázek 11.1: RPC komunikace v Amoebě

Proti napadení se Amoeba brání tajením číslo portu. Každý vstupní bod má přiřazeny dva identifikátory, označované jako **get_port) a put_port. První z nich je privátní, druhý všeobecně známý. Číslo reprezentující put_port vznikne z get_port transformací pomocí jednosměrné funkce. Vyvolá-li server funkci get_request, vypočítá systém put_port a uloží jej do tabulky portů, u kterých naslouchá. Při příjmu zprávy se podle tabulky vypočte get_port, který je tajný a sítí se nikdy nešíří.**

Sémantika RPC je typu maximálně jednou. Jestliže se RPC uskuteční, systém garantuje že RPC nebude přenášeno více než jednou i když server zkrachuje a rychle rebootuje.

Skupinová komunikace **Pro efektivní řešení některých problémů v distribuovaných systémech je vhodné použít model komunikace 1:N.Amoeba poskytuje pro tento způsob komunikace mechanizmus tzv. spolehlivého broadcastu, který zajišťuje, že zprávu poslanou skupině uzlů obdrží všichni. Pokud je takovýchto zpráv vysláno více, jsou do všech uzlů doručeny ve stejném pořadí.**

Spolehlivý broadcast je realizován pomocí tzv. sequenceru, což je proces, který běží na jednom z uzlů systému. Odesílatel zprávy volá pomocí RPC sequencer, sequencer přidá ke zprávě záhlaví, obsahující číslo zprávy a rozešle ji příjemcům. Přijímací vlákno zprávu přijme, zkontroluje její pořadové číslo, a pokud je tato zpráva očekávána, tak ji zpracuje. Je-li očekávána zpráva s jiným pořadovým číslem, odloží ji a požádá sequencer o zaslání chybějících zpráv.

Primitivní operace pro skupinovou komunikaci **Primitivní operace pro skupinovou komunikaci musí obsahovat nejen prostředky pro přenos zpráv, ale i prostředky pro vytváření a rušení skupin. Skupiny v Amoebě jsou uzavřené a členy skupiny jsou procesy.**

Mezi tyto primitiva patří:

- **Create Group - slouží k vytvoření skupiny. Parametry určují změny velikosti a spolehlivost. Vrací identifikátor skupiny.**
- **Join Group - posílá všem členům zprávu, že se do skupiny připojuje další člen.**
- **Leave Group - posílá všem členům zprávu že se ze skupiny odhlašuje člen. Seznam všech členů skupiny se používá tehdy, když je třeba skupinu rekonstruovat. Při opuštění skupiny posledním členem se skupina ruší.**
- **Send To Group - uspořádané posílání zpráv. Zprávy jsou uspořádány podle času. Zajišťuje uspořádaný příjem, není striktně konzistentní.**
- **Receive From Group - přijímá zprávu určenou pro danou skupinu. Není-li zpráva k dispozici zůstane zablokován. Protokol zaručuje, že se žádná zpráva nemůže ztratit. Pokud se zpráva ztratí, obsahuje protokol mechanizmus pro obnovu ztracených zpráv.**
- **Reset Group - specifikuje minimální množství členů, které musí skupina mít. Není-li dost členů, indikuje se chyba.**

Skupinová komunikace odolná proti chybám **Skupinová komunikace musí být zabezpečena proti chybám. Nesmí se stát, aby zprávu určenou pro skupinu procesů některé z nich obdržely a jiné nikoliv.**

Výpadek procesoru se pozná podle toho, že zpráva poslaná do zkrachovaného procesoru není potvrzena. Proces, který zjistí výpadek procesoru ze skupiny vyšle příkaz Reset Group. Obnova probíhá ve dvou fázích.

1. fáze - každý proces je chápán jako koordinátor.

2. fáze - koordinátor vytvoří skupinu a obnoví všechny ostatní procesy.

Rekonstrukce skupiny probíhá následovně. Jeden nebo více procesů pošle zprávu Reset Group. Podle vyššího čísla poslední přijaté zprávy vzájemně určí kdo bude koordinátor. Vybraný uzel (koordinátor) získá od ostatních uzlů nejnovější zprávy, které mu chybí. Pak sde stane sequencerem a pošle všem uzlům ve skupině zprávu, která zpráva je poslední. Ty pak samostatně získají odpovídající kopie a pokračuje se dál.

Pokud se při vytváření skupiny určí, že více než jedna stanice bude udržovat buffer historie, může zastoupit sequencer při jeho krachu. Buffer historie může jednoduše vytvářet tak, že naslouchá na síti.

Buffer historie obsahuje zprávy tak dlouho, dokud není jisté, že jsou zpracovány ve všech uzlech. Požadavek na sequencera obsahuje také potvrzovanou zprávu (její číslo). Sequencer si musí pamatovat všechna potvrzení. Stanice, která nepotřebuje sama vysílat, musí periodicky posílat samostatná potvrzení. Navíc může sám sequencer posílat zprávu Request for Status, kterou si vyžádá potvrzení od ostatních satnic.

Existují dvě metody, jak posílat broadcast zprávy pomocí sequenceru.

- **stanice pošle sequenceru požadavek pomocí RPC. Sequencer požadavek zaznamená do fronty, přiřadí mu pořadové číslo, a když na něj dojde řada, vyšle jej jako broadcast všem členům skupiny. Požadavek uloží do bufferu historie a čeká na potvrzení od členů skupiny. Když se je zpráva potvrzena ode všech, může ji smazat. Stane-li se, že stanice zprávu nedostane, nic se neděje. Teprve při obdržení další v pořadí zprávy zjistí, že jí jedna schází. Pak si musí nejprve vyžádat starou zprávu potvrdit ji.**

- **stanice pošle broadcast zprávu všem stanicím skupiny. Zprávu dostane také sequencer. Ten ji přiřadí pořadové číslo a to pošle jako broadcast všem stanicím. Teprve při přijetí zprávy od sequenceru mohou stanice zapamatovanou zprávu zpracovat. Po zpracování ji musí pochopitelně potvrdit.**

Protokol FLIP **Pro snažší realizaci algoritmů distribuovaného systému Amoeba byl vyvinut speciální síťový protokol FLIP (Fast Local Internet Protocol). Tento protokol je navržen tak, aby poskytoval maximální podporu algoritmům, používaným v distribuovaných systémech. Podporuje spolehlivou komunikaci typu 1:N i RPC.**

Splňuje následující požadavky:

- **transparentnost - zajišťuje optimální výběr přenosové cesty mezi dvěma uzly. Dovede se vyrovnat i s mobilitou uzlů.**

- **skupinová komunikace - zajišťuje efektivní skupinovou komunikaci s využitím podpory nižších vrstev komunikačních protokolů.**

- **bezpečnost - podporuje šifrování dat, rozlišení bezpečných a nejistých komunikačních cest.**

- **správa sítě - dovede se vyrovnat se situacemi, kdy některý z procesorů vypadne, popř. je přerušen některý síťový spoj.**

- **WAN - dovoluje komunikovat nejen v mnohabodových LAN sítích, ale i po spojích sítí WAN. Není však kompatibilní s TCP/IP, což je jeho nevýhoda, která se musí řešit pomocí TCP/IP bran.**

Přenos zpráv v protokolu FLIP probíhá bez navazování spojení a bez potvrzování. Důvodem je co možná nejvíce snížit zatížení sítě i procesů, které spolu komunikují.

Komunikace probíhá mezi tzv. NSAP (Network Service Access Point), jejichž adresa je 64 bitové číslo. NSAP není závislé na fyzickém umístění v síti, může se přemisťovat od uzlu k uzlu a svoji adresu si ponechává. Jeden uzel může mít přiřazeno i více NSAP, typicky i několik pro každý proces. 64 bitů adresy je rozděleno do 256 adresových prostorů po 56 bitech. Nulová adresa představuje broadcast. Komunikující jednotky (procesy) si své NSAP vybírají náhodně v daném adresovém prostoru. To snižuje možnost kolize adres, usnadňje přemístění procesů a znesnadňuje padělání adres.

Spojení fyzické adresy ze sítí je uskutečněno tzv. FLIP boxem (programová vrstva nebo speciální komunikační procesor). Základním modulem FLIP boxu je tzv. packet switch (přepínač paketů), který si dynamicky udržuje směrovací tabulku a slouží k přenosu paketů mezi jednotlivými sítěmi, resp. počítačem a sítí. Jednotlivé sítě jsou k přepínači paketů napojeny pomocí síťového rozhraní, počítač pomocí rozhraní hostitelského počítače.

Rozhraní hostitelského počítače definuje sedm možností volání ve směru uzel - síť a dvě volání v opačném směru. Jejich význam je následující:

- **flip_init - proces si přidělí položku v rozhraní**
- **flip_end - uvolnění položky rozhraní**
- **flip_register - proces registruje NSAP adresu. Registrovaná adresa je privátní adresa procesu. Zveřejněna je veřejná FLIP adresa, která je zakódovanou verzí privátní adresy. To znamená, že proces má na každé straně rozhraní jinou adresu a jiný proces může tomuto procesu zprávy pouze posílat, ale nemůže přijímat zprávy, určené tomuto procesu.**
- **flip_unregister - odstranění položky z tabulky**
- **flip_unicast - poslání zprávy jednomu příjemci**
- **flip_multicast - poslání zprávy N příjemcům**
- **flip_broadcast - poslání zprávy všem příjemcům**

Připojenému uzlu posílá rozhraní následující zprávy:

- **notdeliver - zpráva nemohla být doručena**
- **receive - přijatá zpráva**

Paket protokolu FLIP je tvořen hlavičkou a datovou částí. Hlavička má pevnou část dlouhou 40 slabik část s proměnnou délkou. Nejdůležitější položky pevné části jsou:

- **MaxHopCount, ActHopCount - k počítadlu ActHopCount je v každém FLIP boxu uzlu, který pakety směruje, přičtena váha prošlé sítě. Překročí-li hodnotu MaxHopCount, je ze sítě odstraněn.**
- **Destination Address, Source Address - NSAP adresy příjemce a odesílatele.**
- **Type - označuje jeden ze šesti typů zprávy.**
- **Length - délka daného fragmnetu zprávy.**
- **Total Length, Offset - délka celé zprávy a pozice datového segmnetu v ní.**
- **Flags - příznaky přenosu zprávy. Patří mezi ně:**

  - **Security - zpráva nesmí být přenášena přes nejistou síť.**
  - **Unreachable - pro zprávu, ve které je nastaven bit secirity neexistuje bezpečná cesta.**
  - **Unsafe - zpráva byla doručena přes nejistou síť.**

FLIP provádí směrování na základě výběru nejvhodnější přenosové cesty. Parametry tohoto výběru jsou váha sítě (čím rychlejší síť, tím nižší váha) a bezpečnost sítě. Bezpečnost sítě je příznak, nastavovaný administrátorem sítě.

Každý FLIP box si udržuje směrovací tabulku, která obsahuje následující položky:

- **Address - jedna nebo více NSAP adres**
- **Network - síťové rozhraní, na kterém se NSAP nachází**
- **HopCount - váha cesty k Address**
- **Trusted - cesta k Address je považována za bezpečnou**
- **Age - stáří položky v tabulce. Položka je postupně inkrementována, při každém použití cesty popsané položkou je nulována. Překročí-li určitou mezní hodnotu, je zrušena.**
-

Směrovací tabulky jsou nejprve prázdné. Naplňovány jsou dynamicky při zasílání zpráv. Pokud popsaná cesta přestane existovat, je zrušena díky mechanizmu časového omezení. Nalezení nové cesty se děje pomocí broadcastu LOCATE, který vysílá FLIP box v případě, že je požádán o doručení zprávy na adresu, ke které nezná cestu. FLIP box obsahující danou adresu odpovídá zprávou HEREIS. Broadcasty jsou vysílány s postupně se zvyšující položkou MaxHopCount počínaje hodnotou jedna. Tento mechanizmus dovolí přednostně vyhledat rychlé přenosové cesty.

Další typy paketů jsou:

- **LOCATE - požadavek nalezeni cesty k zadané adrese**
- **HEREIS - odpověď na LOCATE**
- **NOTHERE - cílová adresa nenalezena (cesta je delší než MaxHopeCount nebo neexistuje**
- **UNTRUSTED - paket má nastaven bit Security ale bezpečná cesta neexistuje**
- **UNIDATA - komunikace 1:1**
- **MULTIDATA - komunikace 1 : N**

11.1.7 AMOEBA servery

Složitější funkce, které nejsou zajišťovány mikrojádrem jsou realizovány speciálními procesy, běžícími v režimu uživatel, tzv. servery. Při vytváření serverů byl použit jednotný přístup tzv. objektů a vlastností.

Souborový systém Souborový systém operačního systému Amoeba je realizován jako proces na uživatelské úrovni. Toto řešení má výhodu v tom, že uživatel může bez zásahu do jádra operačního systému nahradit tento proces jiným, a tím změnit celou definici a chování souborového systému. Druhou zajímavou vlastností je oddělení správy souborů a správy adresářů. Souborový server manipuluje pouze se soubory (objekty) na základě identifikátorů (vlastností). Adresářový server mapuje jména souborů na jejich identifikátory (vlastnosti). Operační systém je standardně dodáván se souborovým serverem, nazývaným Bullet server.

Bullet server Bullet server je souborový server navržený s ohledem na maximální rychlost prováděných operací. Některé koncepce, které zavádí jsou neobvyklé v běžném pojetí souborových systémů. Soubory jsou nemodifikovatelné. Soubor může být vytvořen, čten a smazán. Fakt, že soubor nemůže být modifikován umožňuje, aby byl uložen na disku i v operační paměti v jedné souvislé oblasti. Načtení souboru může být provedeno v rámci jedné diskové operace. Uživateli může být poslán pomocí jednoho RPC. Výhoda je vysoká přenosová rychlost, nevýhodou je omezení délky souboru na velikost, která se vejde do paměti jako celek.

Bullet server si udržuje tabulku souborů, která je indexována číslem objektů (souborů), a pro každý soubor obsahuje kontrolní pole pro ověření oprávněnosti přístupu a dvě ukazovátka - jedno ukazuje na místo uložení souboru na disku, druhé na místo uložení souboru ve vyrovnávací paměti.

Soubory uložené na bullet serveru lze rozdělit na uncommited - nedokončené, dočasné, zapisované a commited - ukončené, stálé.

Nad bullet serverem se provádí operace:

Create - vytvoření souboru

Read - čtení souboru nebo jeho části

Size - vrací velikost souboru

Modify - přepis n slabik

Insert - přidání n slabik

Delete - vypuštění n slabik

Directory server **Adresářový server provádí mapování jmen objektů na číselné jednoznačné identifikátory. Objekty - adresáře vytvářené adresářovým serverem na žádost klientů obsahují seznamy dvojic (jméno, seznam vlastností). Vlastnosti mohou odkazovat na libovolné objekty, tedy i na adresáře. To dovoluje vytvářet hierarchické struktury adresářů. Na objekty adresáře mohou být aplikovány operace přidání a smazání položky.**

Každá dvojice (jméno, vlastnosti) může pro každé jméno souboru obsahovat několik vlastností, které odkazují na repliky souboru, uložené na různých souborových serverech v systému. Tato replikace je prováděna automaticky operačním systémem v zájmu zvýšení dostupnosti a bezpečnosti uložení souborů. Automatická replikace je umožněna nemodifikovatelností souborů.

Seznam vlastností pro dané jméno může být rozdělen do několika skupin, přičemž vlastnosti z každé skupiny definují jinou úroveň přístupových práv k souboru. Klient může znát vlastnost identifikující adresář, která mu umožňuje přístup pouze k určité z daných skupin vlastností. Tímto způsobem je možné simulovat systém ochrany souborů podobný UNIXu.

Protože adresářový server poskytuje služby kritické pro běh systému, je provozován současně na více uzlech, přičemž je třeba zajistit konzistentnost dat na jednotlivých uzlech. K tomu se s výhodou používá popsaný mechanizmus spolehlivého broadcastu, pomocí něhož server informuje ostatní adresářové servery o změnách provedených v adresáři.

Položka adresáře má následující strukturu:

ASCII	Capability			Owner	Group	Other

Obrázek 11.2: Struktura položky adresáře

Nad adresářem jsou definovány operace:

create, delete - pro celý adresář

add, delete - pro řádek adresáře

look up - pro prohlížení adresáře

Adresářové služby pro soubory jsou: Create, Delete, Append, Replace, Look up, Get masks a Chmod.

Replication server **Replikace souborů se provádí na pozadí. Říká se jí "líná" replikace, protože se provádí se zpožděním.**

Run server **Run server slouží k rozdělování úloh jednotlivým procesorů procesorového poolu.**

Boot server Boot server zajišťuje odolnost systému Amoeba proti výpadkům. Každý proces, který má "přežít" i výpadek stroje na kterém běží, se zaregistruje voláním služby na boot serveru. Boot server pak zaregistrovaným procesům periodicky vysílá výzvy, na které očekává korektní odpovědi. Neobdrží-li odpověď od některého procesu ani po stanoveném počtu pokusů, spustí tento proces na novém procesoru z procesorového poolu.

TCP/IP server TCP/IP server slouží jako brána pro služby, které používají TCP/IP protokol. Řeší nekompatibilitu FLIP vůči ostatnímu síťovému světu.

11.2 AFS

Andrew File System vznikl jako součást projektu Athena, projektu pro distribuované výpočetní prostředí. AFS je distribuovaný souborový systém. Je komerčně dostupný. Jako zálkad pro DFS (Distributed File System) jej zvolilo OSF (Open Software Foundation).

Cílem při vytváření AFS bylo:

- 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)

Projekt AFS vznikl na Information Technology Center na Carnegie Mellon University ve spolupráci s firmou IBM. Vyzdvihuje potřebu unifikovaného prostředí pro množství různých pracovních stanic propojených heterogenní sítí. Je orientován na výkonné pracovní stanice a operační systém UNIX. 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

11.2.1 Kompatibilita s UNIXem na úrovni jádra operačního systému

AFS zachovává funkčnost unixovských stanic a unifikuje jejich chování tak, že se všechny tváří stejně, nezávisle na konkrétní stanici. Tím se minimalizují nutné zásahy do existujícího aplikačního programového vybavení. Do jádra operačního systému je přidán speciální program, který zajistí odchytávání požadavků na jádro a jejich předání speciálnímu programu - Andrew Cache Manageru. Cache Manager zajistí komunikaci se serverem, přenese celý požadovaný soubor na lokální disk a dá jej k dispozici volajícímu uživatelskému programu.

11.2.2 Manipulace se soubory

Při volání funkce open se celý soubor přenese na stanici, kde se s ním pracuje. Na server se přenáší pouze po uzavření souboru, po volání close nebo sync. Protože se celý soubor přenáší najednou, může dojít u velkých souborů ke zpoždění při prvním přístupu. Soubory, které jsou jen pro čtení se přenesou do vyrovnávací paměti jen jednou. Jejich umístění spravuje Cache Manager. Pokud dojde k výpadku serveru, přejde uzel do lokálního režimu a všechny soubory, které byly umístěny do vyrovnávací paměti jsou nadále k dispozici. Při startu stanice Cache Manager konzultuje platnost všech souborů ve vyrovnávací paměti se serverem. Server eviduje všechny soubory a v případě změny některého z nich na to upozorní Cache Manager stanice.

11.2.3 Struktura serverů

Systém je rozdělen do relativně malých serverů, čímž se dosahuje lepšího poměru výkon/cena a vyšší spolehlivost. Všechny řídicí informace jsou uloženy v distribuovaných databázích. Tyto informační databáze obsahují informace o zdrojích systému a jejich rozmístění. Umístění informačních databází závisí na rozložení ostatních serverů v systému. Pokud je to výhodné, mohou být na stejných strojích jako data, mohou však pracovat i na vyčleněných serverech.

11.2.4 Přístupová práva

Přístup k jednotlivým adresářům je řízen pomocí seznamu přístupových práv (Access Control List). Tento seznam obsahuje dvojice subjekt (uživatel, skupina), přístupová práva. Přístupová práva k adresářům jsou:

- read

- write

- insert

- delete

- lookup

- lock

- administer

Vazba na přístupová práva UNIXu je zachována s tím, že k přístupu je nutno mít kromě přístupových práv UNIXu k souborům také příslušná přístupová práva ACL k adresářům. UNIXová přístupová práva pro skupinu jsou ignorována. Seznam přístupových práv se dědí. Vytvoří-li se nový adresář, zdědí přístupová práva svého předchůdce. UNIXové právo vykonat (execute) samostatně nemá smysl, pro spuštění je nutno mít právo přečíst.

11.2.5 Skupiny (Protection groups)

Každý uživatel může vytvářet vlastní skupiny uživatelů. Označují se $uivatel : skupina$. Existují systémové skupiny jako např. $system : administrators$.

11.2.6 Ověřování

AFS využívá systému ověřování pomocí serverů KERBEROS. Jméno uživatele a jeho heslo je registrováno v ověřovacím serveru. Při ověřování pošle uživatel své jméno a ověřovací server mu pošle vstupenku (TGT - ticket granting ticket) zašifrovanou klíčem, odvozeným od hesla uživatele. Po rozšifrování ji uživatel používá pro komunikaci s dalšími servery AFS. Během této úvodní sekvence dojde k ověření pravosti obou komunikujících stran. Ke komunikaci v AFS se používají tzv. pověření (tokens, forma vstupenky), které se získají na základě TGT vstupenky.

Kerberos dělí oblasti, kde zabezpečuje ověřování na království (realm). AFS rozděluje oblasti na buňky (cells). Jsou to samostatné oblasti z hlediska správy. Pro každou buňku musí získat uživatel zvláštní pověření. Vstupenky mají omezenou platnost. Pověření (token) je vydán pro trojici {stanice, uživatel, skupina procesů}. Stanice je identifikována svojí adresou, uživatel svým UID z UNIXu a skupina procesů číslem identifikujícím skupinu. Označuje se jako PAG (Protection Access Group). Není-li PAG uvedena, je nahražena UID procesu. Superuživatel nemá žádná zvláštní práva. Mechanizmus propůjčení identifikace zrácí smysl a funkčnost.

11.2.7 Souborový systém z pohledu AFS klienta

Soubory souborového systému AFS patří do jednoho jmenného prostoru. AFS jmenný prostor je reprezentován jedním stromem adresářů. Kořenem je zvolen /afs, kam se souborový systém připojí při inicializaci klienta. Na další úrovni jsou připojeny jednotlivé buňky (např. zcu.cz). Dále pokračují adresářové stromy jednotlivých buněk. Celý strom je sestaven připojováním svazků.

11.2.8 Souborový systém z pohledu AFS servera

Z pohledu servera je souborový systém rozdělen do svazků. Svazek je skupina souborů a adresářů logicky k sobě patřících. Z hlediska celého systému souborů je to podstrom, který se připojuje do globálního stromu. Svazek má přiřazen limit své velikosti (quota). Umístění svazku je pro uživatele transparentní. Jeho přístup je automaticky zajištěn prostřednictvím distribuované databáze. Svazky lze přesouvat bez přerušení dostupnosti. Každý svazek, určený pouze pro čtení, může mít několik replik, rozmístěných libovolně na různých serverech. Při mapování se automticky volí nejsnadněji přístupná replika z funkčních. Interně lze repliky vytvářet klonováním svazků. Vytvoří se pouze nová položka v databázi. Jednotlivé soubory jsou postupně přepisovány do nového svazku. Vše se děje na pozadí, není přerušen přístup uživatele k datům, ani odstaven celý systém.Technika se používá pro snadné zálohování. Z pohledu uživatele nejsou (pokud nechce) svazky vidět.

11.2.9 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. Použitelnst 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.

11.3 DCE

Historie DCE OSF (Open System Foundation) byl vytvořen skupinou hlavních výrobců výpočetní techniky včetně IBM, DEC a HP jako odezva na AT&T a SUN. Cílem bylo vyvinout novou verzi UNIXu, kterou by nekontrovalo AT&T a SUN (licenční poplatky). Výsledkem byl operační systém OSF/1. Od začátku bylo jasné, že zákazníci chtějí nad OSF/1 tak jako nad UNIXem budovat distribuované aplikace.

OSF odpovědělo na tyto potřeby vydáním "Request for Technology", ve kterých se ptalo podniků na podpůrné prostředky a ostatní programové vybavení, které potřebují spojit s distribuovaným systémem. Mnoho podniků odpovědělo a OSF na základě těchto odpovědí vytvořilo integrovaný balík - DCE, který může pracovat jak nad OSF/1, tak i nad jinými systémy.

Nyní je DCE jedním z hlavních produktů OSF. Doplňkový produkt, DME (Distributed Management Enviroment), určený pro údržbu distribuovaných systémů je sice plánován, ale nebyl dosud vytvořen.

Struktura NCA **NCA (Network Computing Architecture) specifikuje transportní komunikační cesty mezi klienty a servery. Dovoluje použít heterogenní sítě jako jednu nedělitelnou aplikační doménu. NCA je ale více než obecná specifikace RPC.**

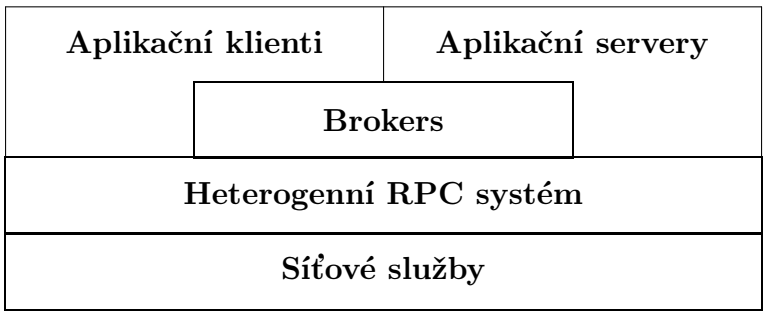
NCS (Network Computing System) je přemistitelná realizace NCA, která může být použita pro vytváření a spouštění distribuovaných aplikací.

NCA je abstrakce, soubor oncepcí a návodů pro distribuované počítání.

NCS je kombinace dvou oddělených produktů.

NCS/NCK **Network Computing Kernel je podpora pro práci v reálném čase. Obsahuje knihovnu RPC a programové vybavení Location Broker. Dále obsahuje podpůrné programy pro administraci síťových služeb založených na NCS.**

NCS/NIDL **je prostředek pro vývoj vlastních aplikací. Tyto aplikace mohou běžet v prostředí vícevláknových procesů, proto jsou programy obsažené v RPC knihovně reentrantní.**



Obrázek 11.3:

DCE je zkratka z Distributed Computing Enviroment. Toto distribuované prostředí bylo vybudováno nad existujícím operačním systémem. Cílem OSF (Open System Fundation) je vytvořit dvě komponenty. DCE a DME (Distributed Management Enviroment). V tuto chvíli je popsána pouze prvá část.

- **NIDL - Network Interface Definition Language**
- **NCS - Network Computing System**
- **DCE - Distributed Computing Enviroment**
- **NCA - Network Computing Architecture**

- **NDR - Network Data Representation**

Distribuované výpočetní prostředí

- podmíněno rychlými sítěmi (LAN i WAN)

- zajišťuje přístup do databází a dalších služeb

- výpočty mohou být rozložené na mnoha uzlech

- umožňuje přístup ke specializovaným procesorům

- zvyšuje dostupnost výpočetních prostředků

- zvyšuje odolnost systému proti poruchám

NCS je realizace z NCA, práce pro všeobecné použití, navržené pro podporu konstrukce distribuovaných programových systémů. Snaží se vyřešit nově vzniklé problémy, spojené s distribuovaným počítáním.

NCA je obecný model pro distribuované počítání. Definuje komunikační protokol pro realizaci rozhraní volání vzdálených podprogramů mezi jednotlivými procesy a distribuovanou aplikací.

NCS je implementace podle tohoto modelu.

managers **NCA počítá s vybudováním knihoven vzdálených podprogramů pro síťové služby. Použití nového managera v distribuované aplikaci je analogické spojení s novou knihovnou do konvenční aplikace.**

NIDL **NIDL je jazyk, který dovoluje definovat rozhraní tak, že procedury mohou být volány pomocí prostředků NCA/RPC.**

Pomocí DCE lze vytvářet aplikace na základě modelu klient - server. Zálkadem DCE je systém volání vzdálených podprogramů (RPC). Zajišťuje komplexní mechanizmus výměny informací včetně lokalizace servera, spojení s ním a provedení volání.

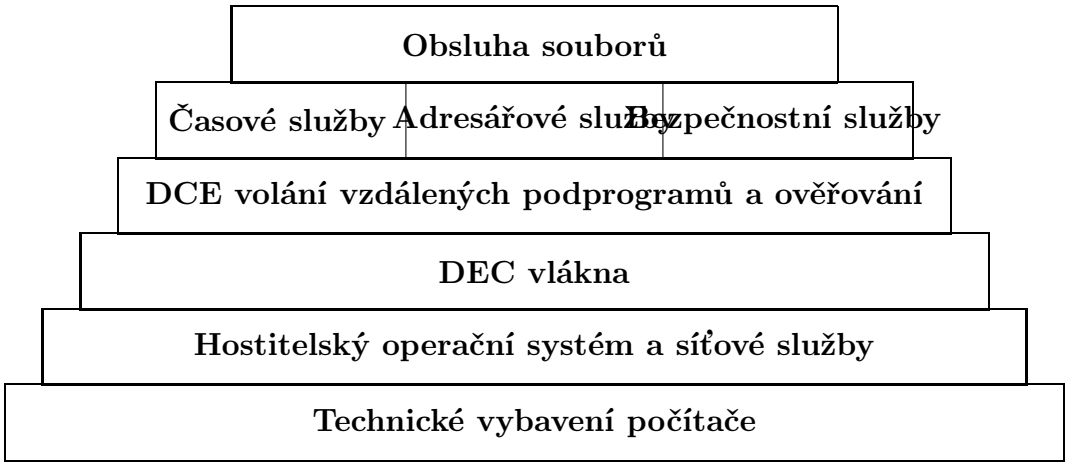
DCE zahrnuje tři základní skupiny služeb, nad kterými je realizován distribuovaný systém souborů. Strukturu ilustruje následující obrázek.

11.3.1 Buňky

Buňky dovolují sdružovat uživatele, počítače a ostatní zdroje. K buňkám jsou vztaženy jména, bezpečnost, administrace a ostatní aspekty.

Buňky se vytváří podle

- **účelu, např. jde o společné dílo skupiny programárorů**
- **bezpečnosti, uvnitř buněk je dána větší důvěra při přístupu ke zdrojům**



Obrázek 11.4:

- **míra režie, DCE je optimalizováno pro lokální prostředí (spolehlivé, rychlé sítě).**
- **administrace, systém je rozdělen do více relativně samostatných celků také z důvodu administrace.**

11.3.2 Vlákna

DCE je vytvářeno s ohledem na možnost zajištění paralelního běhu s minimálními požadavky na režii systému při vytváření paralelních aktivit (procesů). Proto byla využita koncepce vláken, známá i z jiných operačních systémů. Funkce pro práci s vlákny jsou sučástí knihovny, nejsou integrální součástí jádra operačního systému. Základem pro realizaci volání služeb, souvisejících s vlákny je doporučení POSIX P1003.4a.

Rozdíl mezi vlákny a procesy spočívá v tom, že jeden proces může vytvořit několik vláken, která sdílí programový i datový prostor procesu. Při přístupu ke společným proměnným není tedy třeba volat příslušné služby jádra operačního systému, ale manipulovat s proměnnými přímo. Aby bylo možné volat rekursivně podprogramy, je každé vlákno vybaveno vlastním zásobníkem. To též dovoluje přerušit běh vlákna kdekoliv a zapamatovat jeho stav. Případné kolizní situace, které by souběžným asynchronním během vláken mohly vzniknout musí programátor předvídat a v rámci daného procesu je řešit sám.

Stavy vláken **Vlákna se mohou nacházet v následujících stavech:**

- běžící, pokud je právě zpracováváno

- připraveno, pokud je připraveno k činnosti, ale chybí mu procesor

- čekající, pokud čeká na ukončení nějaké operace

- ukončeno, pokud skončilo svoji činnost, ale musí ještě čekat na ukončení procesu potomka

Pokud je vlákno připraveno, čeká ve frontě připravených, která je ovládána některým z následujících algoritmů plánování:

- FIFO, strategie plánování podle pořadí příchodu požadavku. Jedná se o nepreemptivní plánování, vlákno ukončí svoji činnost teprve tehdy, až o to samo požádá.

- cyklická obsluha, kdy vlákno dostane přiděleno časové kvantum a po jeho vyčerpání je přeplánováno.

- implicitní, které využívá strategii cyklické obsluhy s tím, že priorita procesu je dána velikostí časového kvanta, které je mu přiděleno (vyšší priorita, více času).

Synchronizace **K synchronizaci mezi vlákny slouží dva typy synchronizačních prostředků:**

- mutex

- podmíněné proměnné

Mutex složí k uzamykání jednotlivých částí programu. Opět z důvodu nízké režie existují tři způsoby uzamykání:

- rychlé, které neprovádí kontrolu, zda-li již jednou vlákno zámek neuzamklo. Při druhém uzamknutí dochází k uvíznutí.

- rekurzivní, kde uzamykání je řešeno s kontrolou, která při druhém a dalším pokusu o uzamknutí zvýší čítač. Zámek je uvolněn teprve tehdy, je-li počet uzamknutí a odemknutí párový.

- nerekurzivní, které také kontroluje vlákno na opětné použití zámku. Na tuto situaci však reaguje chybou.

Podmíněné proměnné jsou datové struktury, které řídí výlučný přístup ke kritickým oblastem. Nemají však vlastní mechanizmus uzamykání, využívají mutex pro přístup k nim samotným.

Volání vláken Celkově existuje asi 54 primitivních funkcí pro manipulaci s vlákny. Tyto funkce jsou součástí uživatelské knihovny. Mezi zajímavější patří funkce pro manipulaci s vláknem. Jsou to:

- create pro vytvoření vlákna
- exit pro ukončení činnosti vlákna
- join, kterou volá rodič chce-li počkat na ukončení činnosti potomka
- detach, kterou volá rodič, pokud na ukončení činnosti potomka čekat nechce.

11.3.3 DCE RPC

Systém volání vzdálených podprogramů v DCE je obdobou RPC vytvořeného fy XEROX. K popisu aplikace se používá jazyk IDL (Interface Definition Language). Ke generování spojek klienta a servera slouží překladač IDL. Syntaxe IDL jazyka dovoluje generovat spojky jak pro programovací jazyk C, tak i pro Pascal.

Jazyk IDL umožňuje nadefinovat tři typy sémantik volání vzdálených podprogramů. Jsou to

- maximálně jednou
- alespoň jednou, která se používá pro idempotentní procedury
- volání se všeobecnou adresou.

Systém volání vzdálených podprogramů zahrnuje i funkce registrace služby a získání adresy koncového bodu (Service Access Point) volané skupiny podprogramů (číslo portu). Tyto služby jsou obdobou Portmapperu. Navíc obsahuje službu vyhledání servera, který službu poskytuje. Celkové schema komunikace zachycuje následující obrázek.

11.3.4 Časové služby

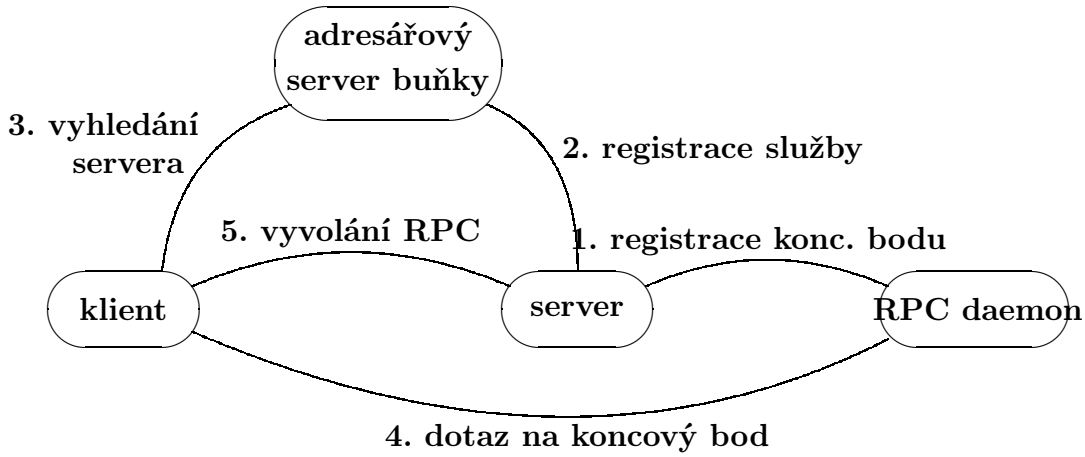
Časové služby jsou zajišťovány pomocí časových serverů. Používá se metoda přenosu časových intervalů. Skutečný čas se odvozuje z průniku časových intervalů časových serverů.

11.3.5 Adresářové služby

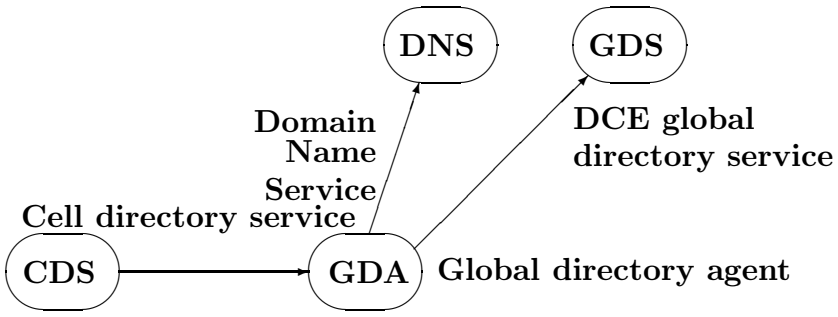
Adresářové služby tvoří významnou část DCE. Je podporováno jak DNS (Domain name system), tak i X.500. Jsou postaveny nad RPC, existuje tedy klient a server. Strukturu volání na straně servera ilustruje následující obrázek.

11.3.6 Distribuovaný souborový systém

Souborový systém byl převzat z AFS. Obsahuje modul pro ochranu přístupu (ACL). Zajišťuje jednotný přístup k souborům.



Obrázek 11.5:



Obrázek 11.6: Adresářové služby