Sun RPC protokol (Remote Procedure Call)

Počítačové sítě
KIV/PSI

Autor

Michal Stašek
A01224
stasekm@volny.cz

Téma

Sun RPC protokol

- Navrhněte program pro zachycování a analýzu protokolu SUN RPC
- Vyberte si některé z provozovaných služeb nad RPC a znázorněte obsah jimi přenášených zpráv
- Pro ověření funkčnosti analyzátoru realizujte jednoduchou dvojici server/klient, která si bude předávat data různých datových typů

Vznik

- firma Sun Microsystems zveřejnila konkrétní řešení mechanismu RPC a jeho standardizaci
- ujalo se a stalo se všeobecně uznávaným standardem v rámci rodiny protokolů TCP/IP (kodifikovaným ve formě dokumentu RFC (Request for Comments)

Důvody vzniku

- vznikl jako mechanismus pro praktickou implementaci NFS (Network File System) – specifikace protokolu RFC 1094

- NFS umožňuje komunikaci klient-server mezi různými platformami – bezestavový (řeší pouze indempotentní operace)

Praktické fungování

základní představa

klient zformuluje požadavek na přístup ke vzdálenému souboru ve formě zprávy, ta je odeslána serveru

|

server zprávu přijme, provede požadované akce

|

výsledek vrátí zpět klientovi jako svou odpověď

Praktické fungování

konkrétní realizace

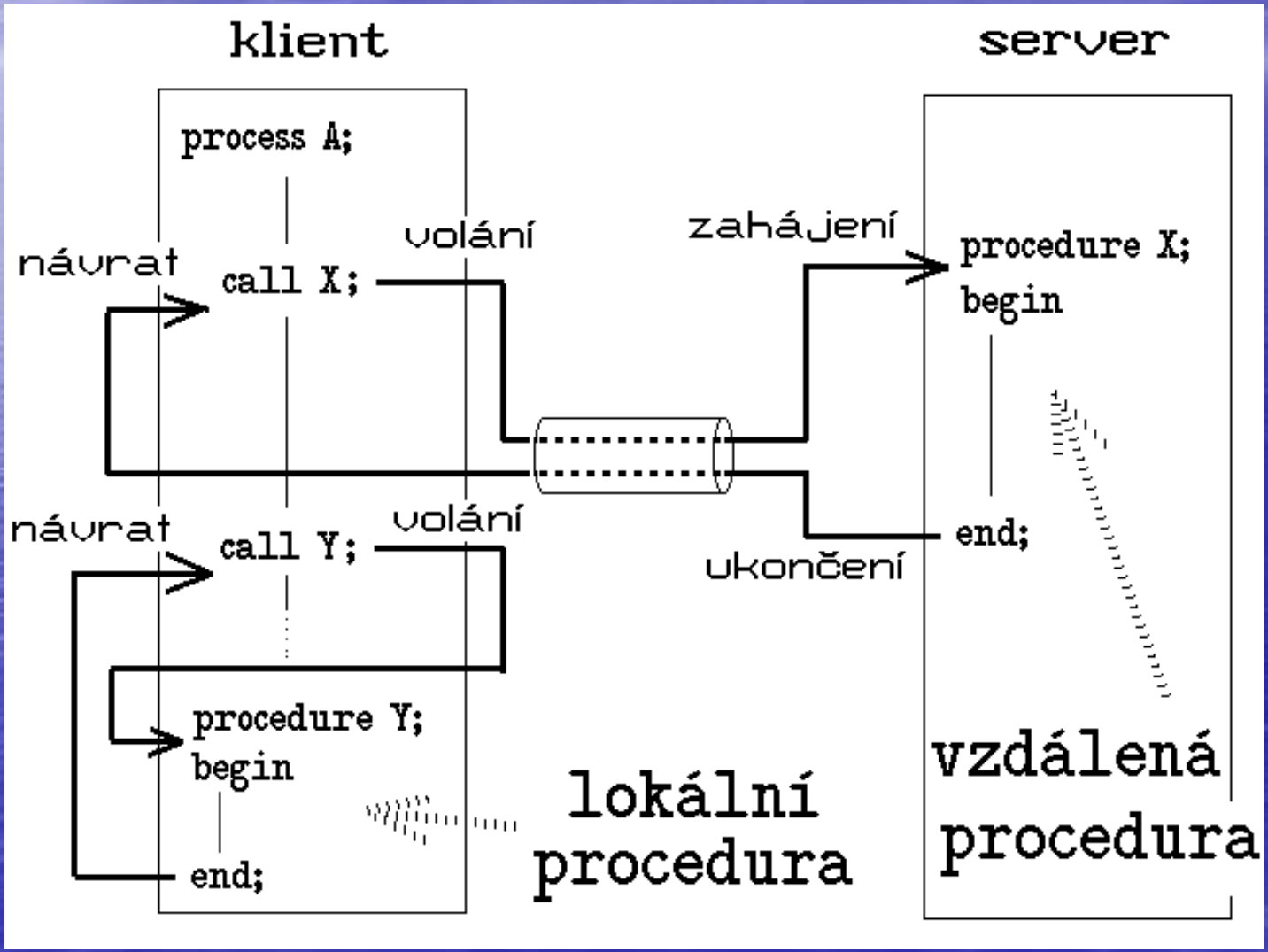
- plně transparentnímu sdílení - aplikace klienta si nemusí uvědomovat, že jde o vzdálený soubor, tj. - svou žádost tedy formuluje přesně stejně, jako kdyby šlo o místní soubor
  pozn. v prostředí Unixu formou systémového volání)

- zvláštní entita, která příslušný požadavek skutečně vyřizuje, ve víceúlohovém prostředí samostatný proces sestaví příslušný požadavek ve formě zprávy, tu odešle a čeká na odpověď serveru

- toto čekání je realizováno jako suspendování příslušného procesu (tedy jeho převedení ze stavu "probíhající" do stavu "čekající"), s požadavkem na následné aktivování v okamžiku příchodu odpovědi (která bude zřejmě signalizována prostřednictvím mechanismu přerušení).

Praktické fungování

představa volání lokální a vzdálené procedury

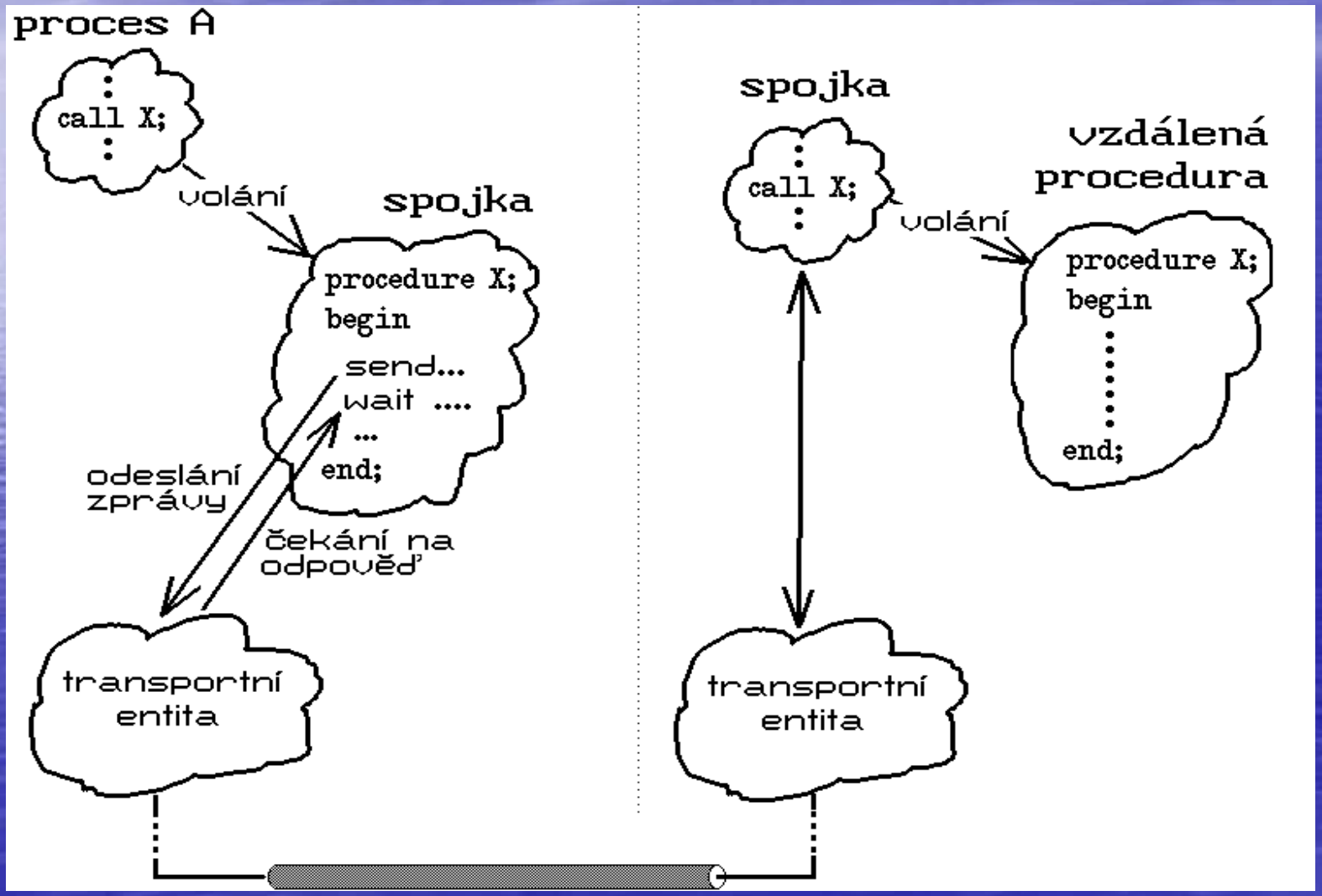


Realizace

- vlastní komunikace mechanismem RPC realizováno v rámci programových entit
- tyto entity vystupují jako lokální procedury, které klient může obvyklým způsobem volat
- proces A, který potřebuje zajistit provedení určitých akcí na vzdáleném uzlu (serveru), volá příslušnou lokální proceduru, která je součástí implementace mechanismu RPC (a označuje se jako **stub**, v češtině pak: **spojka**).
- tato spojka (procedura) zajistí vše potřebné (včetně čekání na příchod odpovědi), a poté řádným způsobem skončí, neboli vrátí řízení zpět procesu A, bezprostředně za místo svého volání.

Realizace

schéma



Definice RPC

RPC definuje:

- spojky na straně klienta
- spojky na straně serveru
  - přijímají zprávy od spojek klientů, a na jejich základě pak volají výkonné procedury, které zajistí provedení požadovaných akcí - tedy procedury, které jsou z pohledu klienta (procesu A) vzdálené, ale pro spojku na straně serveru již jsou lokální!
- dále je součástí definice mechanismu RPC i přesný způsob komunikace mezi spojkami klientů a serverů

Přenos parametrů „marshalling"

- proces A na straně klienta potřebuje načíst část souboru -> volá proceduru (např. *read*), která je ve skutečnosti spojkou (stub)
- spojce-proceduře přitom předá všechny potřebné parametry -> spojka sestaví zprávu pro svou partnerskou spojku na straně serveru, a v ní mj. uvede, která vzdálená procedura má být provedena
- parametry, které spojka klienta dostala při svém volání "patří" vzdálené proceduře => spojka klienta je proto převede do takového tvaru, který je vhodný pro přenos (tomuto úkonu se říká **marshalling**, nebo též: **serializing**), a připojí je ke zprávě, odesílané spojce serveru
- spojka serveru zprávu přijme, parametry "rozbalí" (tzv. **unmarshalling**, **deserializing**), a zajistí volání požadované procedury
- jakmile výkonná procedura skončí, vrátí řízení tomu, kdo ji volal - tedy spojce serveru -> odeslání zpět spojce klienta.

Identifikátory vzdálených procedur

- každá vzdálená procedura má přiřazen jednoznačný číselný identifikátor, a její vzdálené volání (tj. volání na straně klienta) má obecně tvar
  CALL (<číslo_vzdálené_procedury>)

- nutnost vhodného řádu pro přidělování číselných identifikátorů - aby byla zachována konzistence číslování procedur a identifikátory byly skutečně jednoznačné

- nutnost existence jediného centrálního subjektu, koordinujícího přidělování -> řešení Sun Microsystems

Identifikátory vzdálených procedur
pokračování

- přidělovat jednoznačného identifikátoru pro každou vzdálenou proceduru organizačně neúnosné
  -> použití takových identifikátorů, které se skládají ze tří složek:

  - z tzv. čísla programu (**program number**), které souhrnně identifikuje skupinu procedur, zajišťujících určitou službu. Například všechny vzdálené procedury, které jsou používány v rámci implementace protokolu NFS, tvoří jednu takovouto skupinu, a mají tudíž přiřazeno jedno číslo programu (pro NFS konkrétně 10003).
  - z čísla procedury (**procedure number**), které jednoznačně identifikuje příslušnou proceduru v rámci její skupiny,
  - z čísla verze (**version number**)

Parametry vzdálených procedur

- konvence - všechny vzdálené procedury mají jeden vstupní a jeden výstupní parametr
- více parametrů se vloží do datové struktury
- vzdálené proceduře předán ukazatel na tuto datovou strukturu (musí být v přenesena na server)

Reprezentace přenášených dat

- aby obě strany také správně interpretovaly každou jednotlivou část vstupních a výstupních dat vzdálených procedur je nutná určitá konvence

  Př. Budou-li například srozuměny s tím, že obsah dvou bytů má představovat celé číslo bez znaménka, mohou jej stále ještě interpretovat různě - jedna strana může považovat za vyšší ten z obou bytů, který druhá strana naopak považuje za nižší.

- řešení problém správné interpretace přenášených dat
  - > má dvě principiální řešení:
    - každá zúčastněná strana bude předem znát konvence druhé strany
    - zavedení společného mezitvaru =>
      - nezbytné konverze nutné provádět dvakrát (u příjemce i odesilatele)
      + každá strana vystačí vždy jen s jednou sadou konverzních prostředků, nemusí se jakkoli přizpůsobovat případným novým konvencím druhých stran

- toto řešení zvolila firma Sun pro implementaci svého mechanismu RPC

XDR - eXternal Data Representation

- standard XDR definuje jednotný způsob reprezentace přenášených dat, nezávislý na konkrétní architektuře jejich odesilatele i příjemce.

  - zahrnuty konkrétní konverzní rutiny, které mají nejčastěji formu knihovních rutin – tzv. XDR filtry.

- konkrétní realizací této volby je pak standard XDR (eXternal Data Representation), který byl opět zveřejněn, je všeobecně uznáván jako standard v rámci rodiny protokolů TCP/IP, a je kodifikován formou dokumentu RFC.

RPC - strana klienta

- k volání vzdálených procedur stačí díky jednozančné identifikaci jediný prostředek - systémová rutina, pojmenovaná příznačně **callrpc**

- ta má celkem osm parametrů:
  - identifikaci uzlu, na kterém má být vzdálená procedura provedena
  - číslo programu (program number) vzdálené procedury
  - číslo verze (version number) vzdálené procedury
  - číslo vzdálené procedury v rámci její skupiny (procedure number)
  - vstupní parametr vzdálené procedury
  - XDR filtr vstupního parametru (který definuje konverzní rutinu pro převod vstupního parametru do přenosového tvaru)
  - výstupní parametr vzdálené procedury
  - XDR filtr výstupního parametru

RPC – strana serveru

- RPC dispečer (RPC library dispatcher) – obsahuje přehled o všech vzdálených procedurách, které je možné na daném serveru volat

- Funkce:
  - je příjemcem žádostí klientů o volání vzdálených procedur
  - volá ty lokální rutiny, které vzdálené procedury implementují (vystupuje v roli spojky serveru)

- Konkrétním prostředkem, kterým se lokální procedura registruje u RPC dispečera, je systémová rutina **registerrpc** s následujícími parametry:
  - číslo programu (program number) vzdálené procedury
  - číslo verze (version number) vzdálené procedury
  - číslo vzdálené procedury v rámci její skupiny (procedure number)
  - vstupní bod lokální procedury, která implementuje vzdálenou proceduru
  - Vstupní parametr vzdálené procedury
  - XDR filtr vstupního parametru
  - výstupní parametr vzdálené procedury
  - XDR filtr výstupního parametru

Tři úrovně RPC

- mechanismus RPC lze využívat na třech úrovních, zatím jsem popsali prostřední
- ta zná distribuovaného prostředí (tj. existenci vzdálených uzlů)
- vyžaduje také znalost konkrétních vzdálených procedur
- jinak snaha o maximální jednoduchost využití celého mechanismu RPC

Tři úrovně RPC - nejnižší

- nejnižší úroveň řeší např.:
  - jaký konkrétní transportní mechanismus je využíván pro skutečný přenos v síti (zda jde např. o nespolehlivou datagramovou službu protokolu UDP, nebo o spolehlivou spojovanou službu protokolu TCP)
  - jaké jsou časové limity (timeout-y)
  - jak je řešena otázka chyb
  - ověřování přístupových práv a totožnosti (authentication) apod.
- standard RPC je řešen nezávisle na transportním protokolu (implementace nad různými protokoly)
- aplikace, využívající RPC musí znát použitý transportní mechanismus a podle toho koncipovat např. otázku spolehlivosti
- pokud chceme ovlivnit transportní prostředky RPC řeší je na této nejnižší úrovni práce s mechanismem RPC (realizuje prostředky typu **callrpc** pomocí prostředků nižší úrovně) ->značně netriviální – odborníci
- úkol odborníků usnadňuje překladač **rpcgen**, vyvinutý firmou Sun ->na základě obecnějšího popisu ve zvláštním RPC jazyku (velmi blízkému k jazyku C) generuje to, co by jinak bylo třeba explicitně naprogramovat na nejnižší úrovni (ve formě zdrojových textů jazyka C).

Tři úrovně RPC - nejnižší

- na nejvyšší úrovni je celý mechanismus RPC obvykle "zabalen", podstata RPC již nemusí být vůbec patrná -> pouhé volání lokálních procedur
- netřeba dodržovat konvence o jednom vstupním a výstupním parametru
- forma zdrojových knihoven, a mohou být přímo začleněny do zdrojových tvarů nejrůznějších aplikací, psaných např. v C
- tato úroveň určena pro méně náročné aplikační programování, které je ale možné prakticky i bez jakéhokoli tušení o existenci mechanismu RPC

RPC v moderních aplikacích

- s nástupem OOP rostla složitost předávaných dat, parametry a návratovými hodnotami metod mohou být nejen jednoduché datové typy, ale i komplexní struktury a objekty

|

- nové způsoby vzdáleného volání služeb; pro jazyk Java bylo vyvinuto vzdálené vyvolávání metod (Remote Method Invocation - RMI)

|

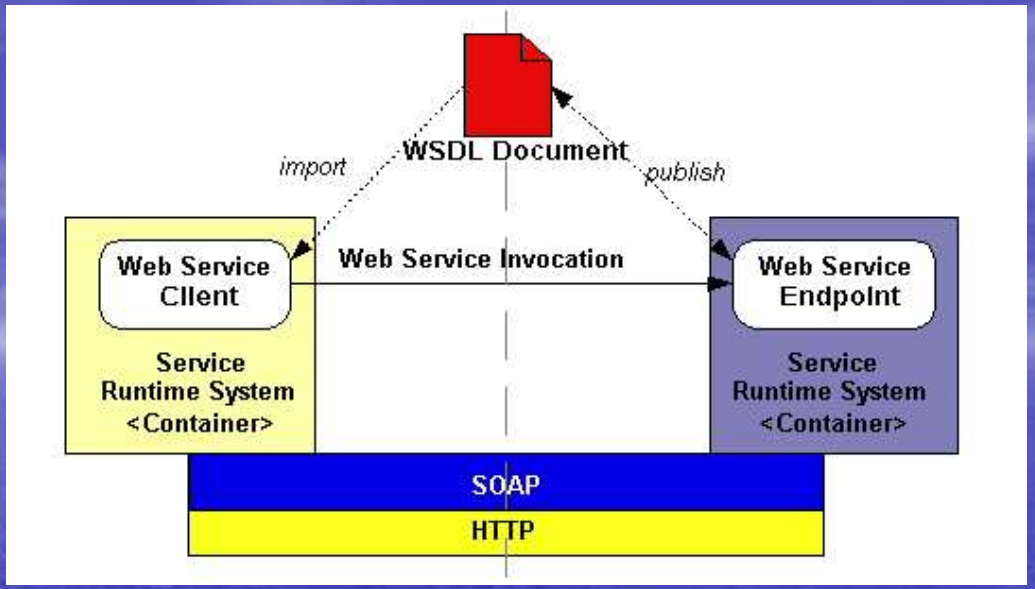
- i RMI má své nedostatky - málo výkonné a hlavně spolupracuje pouze s Javou

|

- Sun znovuoživil RPC a potíž s potřebou přenášet komplexní struktury a objekty vyřešil jejich zakódováním do moderního čistě textového formátu XML

- JAX-RPC je jen jedním kamínkem v mozaice nové otevřené architektury RPC

JAX-RPC



- za zkratkou JAX-RPC se skrývá java API, umožňující vytvářet webové služby a přistupovat k nim prostřednictvím RPC založeného na značkovacím jazyce XML
- toto rozhraní je obálkou protokolu, který zajišťuje vlastní RPC volání a pod ním ležícího transportního protokolu
- aktuální vydání specifikace JAX-RPC (verze 1.1) stanoví, že každá jeho implementace musí podporovat jako RPC protokol SOAP 1.1 a jako transportní protokol HTTP 1.1.
- API zajistí, že vývoj klientské i serverové části aplikace je nezávislý na konkrétních použitých jiných protokolech
- pro usnadnění práce vývojářů JAX-RPC obsahuje vestavěnou podporu pro vzájemné mapování mezi Javou a WSDL
- Webové služby poskytované serverem lze jednoduše popsat jazykem WSDL a urychlit tak implementaci klientské části aplikace

RPC Port mapper

- **rpc.portmap** - zajišťuje svázání jednotlivých služeb rpc s tcp nebo udp porty
- protože jsou porty pro rpc služby alokovány dynamicky, neexistuje jednoznačná korespondence mezi službou a tcp nebo udp portem, na kterém daná služba běží
- Navázání spojení klient-server:

  - RPC klient se tedy před zahájením každé komunikace spojí se službou rpc.portmap na serveru

  |

  - od ní získá na základě čísla serveru informaci o portu, na kterém je server dostupný

Pozn. Ke správné funkci RPC služeb je tedy potřeba, aby se jako první spustila služba rpc.portmap a teprve potom ostatní služby.

rpcinfo

- rpcinfo je služba umožňující získat informace o rpc službách běžících na daném stroji
- **rpcinfo -p host_name**
  - vrací informace o službách běžících na stroji *host_name*

- **rpcinfo -n port_num -t host_name program_name**
  - vrací informace o službě běžící na stroji *host_name* na tcp portu číslo *port_num*

- Příklad výpisu příkazu **rpcinfo -p localhost**

- program verz proto port
  100000 2 tcp 111 portmapper
  100000 2 udp 111 portmapper
  100024 1 udp 32768 status
  100024 1 tcp 32768 status

Fungování v Linuxu

- k tomu, aby Linux umožnil sdílení dat, používá kombinaci podpory v jádře a několika démonů (podporu nfs zakompilována v jádře systému)
- musí být zprovozněnou službu rpc.portmap na všech úrovních běhu systému, kde chceme používat NFS
- mimo rpc.portmap používá NFS následující démony:
  - **rpc.mountd** - proces na straně serveru, spouštějící procesy, které přijímají žádosti o připojení svazku (*mount*)
  - **rpc.nfsd** - proces na straně serveru, implmentující uživatelskou část NFS spolupracující s jádrem na vyřízení dynamických požadavků NFS
  - **rpc.lockd** - démon, který v minulosti zajišťoval zamikání souborů (dnes starost jádra)
  - **rpc.statd** - démon implementující *Network status monitor* RPC protokol; zprostředkovává klientům informace o restartu NFS serveru.
  - **rpc.rquotad** - RPC server, který zprostředkovává informace o právech připojených uživatelů
- minimální instalace musí obsahovat minimálně **rpb.mountd**, **rpc.nfsd** a **rpc.portmap**

Konfigurace serveru

- všechny informace potřebné pro NFS ohledně svazků, které mohou být exportovány, a jejich práv jsou uloženy v souboru **/etc/exports**
- informace z tohoto souboru načítá program **exportfs**, který je předává démonům **rpc.mountd** a **rpc.nfsd**.
- Příkaz **exportfs** nám umožňuje dynamicky přidávat a odebírat svazky nebo adresáře, které se mají sdílet přes nfs, bez nutnosti restartování NFS služeb a poznamenívá tyto informace do souboru **/var/lib/nfs/xtab**
- všechny změny se promítnou do fungování systému okamžitě, protože služby NFS čtou soubor **/var/lib/nfs/xtab** při každém požadavku o přístu k souboru nebo svazku.
- Příkaz **exportfs** má nasledující volby:

  -r provede nové načtení souboru /etc/exports a vytvoření /var/lib/nfs/xtab. Tím dojde k načtení změn provedených v souboru /etc/export.

  -a způsobí, že budou nebo nebudou exportovány všechny adresáře v závislosti na /etc/exports

  -o *options* umožňuje uživateli přidat volbu, která se v souboru /etc/exports nevyskytuje. Tato volba se používá pro testování nových nastavení předtím než jsou zapsána natrvalo do souboru /etc/exports.

  -i tato volba říká programu **exportfs**, aby ignoroval soubor /etc/exports a použil pouze nastavení z příkazové řádky.

  -u zakazuje export zadaného adresáře. (Příkaz **exportfs -ua** způsobí okamžitý zákaz sdílení)

  -v vypisuje informace o exportovaných a neexportovaných adresářích.

- Jesliže spustíme příkaz **exportfs** bez parametrů, obdržíme seznam aktuálně exportovaných adresářů.

Konfigurace klienta

- Každý svazek, který server umožňuje sdílet, může být připojen (mount) několika způsoby:
- příkazem **mount** (musí provést root při každém startu systému a nesmí zapomenout tyto svazky při vypínání nebo restartu odmontovat).
- úpravou souboru **/etc/fstab**. Při startu systému skript */etc/rc.d/init.d/netfs* provede přimontování nfs svazků podle */etc/fstab*.
- použitím systému **autofs**

Použitá literatura

- e-archiv Jiřího Peterky: Počítačové sítě
  www.earchiv.cz
- Fedor Kállay, Peter Peniak: Počítačové sítě LAN/MAN/WAN a jejich aplikace