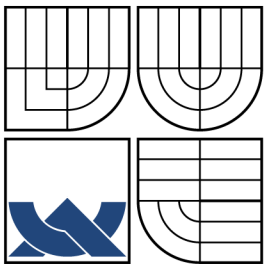
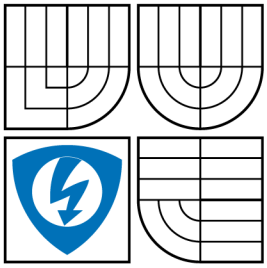


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SKRIPTY PRO KONFIGURACI OPERAČNÍHO SYSTÉMU FEDORA V SÍTI PLANETLAB

SCRIPTS FOR CONFIGURATION OF FEDORA OPERATING SYSTEM IN PLANETLAB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ CZERNER
AUTHOR

VEDOUCÍ PRÁCE — ING. DAN KOMOSNÝ, PH.D.
SUPERVISOR

BRNO 2008

ZDE VLOŽIT LIST ZADÁNÍ

Z důvodu správného číslování stránek

ZDE VLOŽIT PRVNÍ LIST LICENČNÍ SMOUVY

Z důvodu správného číslování stránek

ZDE VLOŽIT DRUHÝ LIST LICENČNÍ SMOUVY

Z důvodu správného číslování stránek

ABSTRAKT

Dokument obsahuje popis nástrojů a postupů pro tvoru skriptů pro vzdálenou konfiguraci operačního systému Fedora, obecně však jakéhokoli unix-like operačního systému. Samozřejmě jsou zde také předvedeny některé způsoby jejich použití a popsána jejich funkce.

KLÍČOVÁ SLOVA

Linux, Fedora, Bash, Planetlab, IPTV, Skripty

ABSTRACT

This document describes tools and techniques for creating scripts for remote configuration of Fedora operating system, generally any unix-like operation system. Of course there is demonstration of using these scripts as well as description of functionality.

KEYWORDS

Linux, Fedora, Bash, Planetlab, IPTV, Scripts

CZERNER L. *Skripty pro konfiguraci operačního systému Fedora v síti PlanetLab*. Místo: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008. Počet stran 76 s., Počet stran příloh 12 s. příloh. Bakalářská práce. Vedoucí práce byl Ing. Dan KOMOSNÝ, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Skripty pro konfiguraci operačního systému Fedora v síti PlanetLab jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne . . . .

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce panu Ing. Danu Komosnému, Ph.D. za podnětné rady a připomínky k vypracování této práce.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tento dokument je bakalářskou prací na téma *skripty pro konfiguraci operačního systému Fedora v síti Planetlab*. Tato práce je součástí většího projektu IPTV Research Group, zabývajícího se ,ve zkratce, především vývojem nových protokolů a software pro celosvětové vysílání internetové televize, tedy potenciálně pro velmi široký okruh koncových uživatelů. Je zřejmé, že projekt, který je v konečné fázi zaměřen na tak velký počet uživatelů je třeba testovat v podmínkách, alespoň se blížících podmínkám reálným. Proto probíhá vývoj a testování v celosvětové síti Planetlab. A zde přichází na řadu má úloha.

Mým cílem bude, jak zadání napovídá, je vytvořit skripty, resp. nástroje, kterými bude možno konfigurovat předinstalovaný operační systém Fedora v síti Planetlab. Půjde o vytvoření skriptů pro příkazový interpret Bash [1], kterými bude možno pomocí stávajících programů jako jsou například ssh [2], nebo scp [3] provádět na jednotlivých uzlech, popřípadě na vybrané skupině uzlů sítě Planetlab operace jako spouštění vlastních skriptů, získávání informací o daných uzlech, instalace nového software, konfigurace programů apod. Vzhledem k tomu, že skripty budou potenciálně využívat i ostatní účastníci projektu, bude nutné je koncipovat tak, aby byly přehledné, s vestavěnou nápovědou a aby v případě nutné změny nebyl pro kohokoli problém si zdrojový kód skriptu modifikovat. Bude tedy nutno skripty také dobře okomentovat.

Mým dalším cílem bude vytvoření nástrojů, které budou sloužit ke sledování "zdraví" jednotlivých uzlů v Planetlabu, tedy jejich dostupnosti a spolehlivosti a prezentace těchto výsledků ve formě webových stránek s grafy popřípadě dalšími informacemi o jednotlivých uzlech.

Tento dokument bude tématicky rozdělen na dvě části a to část teoretických poznatků a část praktického vypracování a ukázek. V teoretické části uvedu do kontextu nástroje, programy a obecně technologie, které jsem ke splnění zadání použil. Samozřejmě také hlouběji popíšu fungování Planetlabu jako celku a zmíním se také o operačním systému Fedora resp. unixových systémech obecně.

V praktické části pak uvedu k čemu dané nástroje slouží, předvedu způsoby jejich použití a popřípadě uvedu co se od mé poslední práce v těchto nástrojích změnilo. Taktéž se zmíním o tvorbě webové preyentace výsledků monitoringu.

[1] Bourne Again SHell – Bournův příkazový interpret.

[2] Secure Shell – program (klient), který implemntuje protokol SSH.

[3] Secure Copy – program (klient), který implmentuje protokol SCP.

1 CÍLE PROJEKTU

Mým cílem v tomto projektu *Skripty pro konfiguraci operačního systému Fedora v síti PlanetLab* bude připravit pro ostatní řešitele projektu skripty, které jim usnadní práci resp. konfiguraci jednotlivých uzlů sítě Planetlab a tím jim umožní zabývat se řešením vlastních problémů, týkajících se zadání jejich práce. Tyto nástroje budou představovat dva skripty pro příkazový interpret Bash. První z nich, již zmíněný v mé předchozí práci *Skripty pro instalaci a konfiguraci operačního systému fedora*, slouží k hromadnému kopírování souborů z resp. na vybrané uzly sítě Planetlab. Od mé předchozí práce nedošlo v tomto skriptu žádným rozsáhlejším úpravám.

Druhý skript pak bude sloužit ke spouštění jakýchkoli programů nainstalovaných na vzdálených operačních systémech v síti Planetlab, stejně jako jakýchkoli skriptů a to jak jednorázových (konfiguračních) tak dlouhodobě běžících vykonávajících nějakou činnost. Základem tohoto skriptu bude program ssh z balíčku programů OpenSSH [1]. Pro tento nástroj mám v plánu připravit i sadu jednoduchých tzv. jednorázových, chcete-li konfiguračních skriptů, jejichž jediným cílem bude nějaký jednoduchý zásah do konfigurace operačního systému Fedora nebo spuštění známých služeb jako je např. crond [2]. Touto sadou ušetřím uživatelům práci s psaním těchto jednoduchých pomocných skriptů, bude tedy okamžitě připraven k použití.

Pro oba zmíněné nástroje bude samozřejmostí vestavěná nápověda a dobře okomentovaný zdrojový kód, do kterého bude v případě změny či pochopení funkce skriptu nutno nahlédnout, avšak nepředpokládám, že by se tato potřeba měla vyskytovat nějak často, pokud vůbec.

Mým, již dříve zmiňovaným, druhým cílem bude monitorování stavu Planetlabu a jednotlivých uzlů a prezentace výsledků ve formě webových stránek. První část jsem již naznačil v mé předchozí práci, avšak od té doby se mnohé změnilo. Vyvstala nutnost změnit způsob ukládání výsledku monitoringu, jelikož původní jednoduché ukládání do systému adresářů na disk, se ukázalo jako neefektivní z hlediska tvorby statistik resp. generování údajů o jednotlivých uzlech či o celé síti. Jak jsem již naznačil v mé předchozí práci získání takovýchto údajů z velkého množství souborů a ještě většího množství řádků v těchto souborech, bylo pomocí skriptu pro příkazový interpret Bash velmi časově náročné. Možné varianty se změnou skriptovacího jazyka jsem zamítl a obrátil jsem se, dle mého názoru, k jedinému vhodnému řešení – databázi.

[1] OpenBSD Secure Shell – soubor programů zprostředkovávajících spojení pomocí protokolu SSH.
[2] Deamon pro spouštění naplánovaných úloh.

Z výše uvedeného tedy vyplývá potřeba pozměnit původní skript generující statistiky tak, aby byl schopen data ukládat do databáze, v mém případě padla volba na MySQL, jelikož se jedná o databázový systém, který je k dispozici (mimo jiné) pod licencí GPL [3] a je velmi rozšířený, hlavně co se týče spojení LAMP [4]. Samozřejmě se nabízejí i další alternativy jako např. PostgreSQL, ale tento systém již nabízí i funkcionalitu, kterou v tomto projektu ani nevyužiji.

Co se týče grafů, budu je vytvářet pomocí programu gnuplot, jak jsem se již zmínil v mé předchozí práci. Tato část tedy zůstává v podstatě nezměněná. Do webových stránek budu grafy vkládat dynamicky pomocí PHP [5].

[3] GNU General Public Licence – všeobecná veřejná licence GNU.

[4] Kombinace Linux + Apache + MySQL + PHP (Perl, Python) používaná pro webový server.

[5] Rekurzivní akronym z PHP: Hypertext preprocesor – PHP: hypertextový preprocesor; Jde o skriptovací programovací jazyk.

2 TEORETICKÉ POZNATKY

Při tvorbě této práce jsem použil spousty nástrojů, programů, obecně technologií, které bych rád předem přiblížil, než začnu mluvit o jejich použití. V prvé řadě stojí operační systém, kolem kterého se celá tato práce točí. Je to operační systém Fedora. Bohužel toto není zrovna přesné, jde o distribuci operačního systému GNU/Linux [1] (dále jen linux) z dílny firmy RedHat, která nese název Fedora. Linux se dá zařadit jako tzv. unix-like [2] operační systém. Už z tohoto pojmenování vyplývá, že tento systém má cosi společného s operačním systémem Unix.

2.1 Unixové operační systémy

2.1.1 Historie

Historický vývoj Unixu byl velmi zajímavý a dá se říci, že ho ve velké míře doprovázela náhoda, jak už to u takovýchto velkých projektů bývá. Když v roce 1965 založilo seskupení firem MIT, AT&T a GE (General Electric) v Bellových laboratořích projekt Mulics, nikdo netušil jak nepoužitelný, ale zároveň historicky významný operační systém vyvinou. První část jim došla celkem rychle, když byl v roce 1969 projekt zastaven z obav že náklady na jeho vývoj se stanou neúnosnými. A důvod? Jednoduše řečeno, systém se stal příliš složitým.

V Bellových laboratořích však zůstala skupinka nadšenců, kteří se nového operačního systému nehodlali vzdát a začali se poohlížet po různých alternativách. Byli ochotní zkusit něco dalšího. Jenže co ? Zatímco se debatovalo o tom *co* dělat, pohrával si Ken Thompson s programováním hry Space Travel, kterou pak portoval na počítač PDP-7 [3]. Jak se ukázalo, způsob jakým k portaci došlo, se stal pro vývoj Unixu klíčovým. Hra byla původně napsána v GEMAP assembleru pro cross-assembler, který pracoval pod GECOS [4], byl však schopný produkovat i pásky pro PDP-7. Stejnou metodou pak byl psán základ Unixu. Systém tedy vznikal na jiném počítači, než na jakém pak běžel.

Po hře pak v roce 1969 následoval filesystém, který byl napsán Thomsonem. Tak postupně začal vznikat operační systém, který dostal jméno Unix. Základní myšlenkou tohoto nově vznikajícího systému bylo, že jádro systému bylo odděleno od

[1] GNU/Linux – jediný správný název tohoto operačního systému, jelikož Linux samotný je pouze název jádra. Z hnutí GNU pak pochází překladač a většina systémových komponent. Dnes je ovšem většinou nazýván jen linux.

[2] Operační systém unixového typu – takový operační systém, který se snaží o co největší podobnost s Unixem a jeho filosofií. Např. GNU/Linux, FreeBSD, či dokonce Mac OS X.

[3] Minipočítač od firmy Digital Equipment Corporation.

[4] General Comprehensive Operating System.

příkazového interpretu, který byl zahrnut do uživatelského prostoru. Tato myšlenka byla částečně převzata Multicsu.

První verze Unixu byla napsána v assembleru, ale postupem času se projevila potřeba pokračovat v tvorbě Unixu v nějakém vyšším programovacím jazyce. Nejprve Thomson zkoušel Fortran, ale to po chvíli vzdal a začal pracovat na vlastním programovacím jazyku, který nazval "B". Tento jazyk měl ale mnoho problému, nehledě na to že byl interpretovaný a tedy pomalý. Thomsonův kolega Dennis Ritchie přiložil ruku k dílu a z původního jazyka "B" se stala vlastně první fáze jazyka "C" jak ho známe dnes.

Kernel Unixu byl pak přepsán do *céčka* a další vývoj pokračoval v tomto programovacím jazyce. A díky tomuto kroku získali v podstatě na platformě nezávislý operační systém. To byla v té době v podstatě revoluce, jelikož do té doby bylo v podstatě nemyslitelné, aby na dvou rozdílných strojích běžel stejný operační systém.

V téže době se ale vyřešil ještě jeden závažný problém Unixu a tedy, jak předávat data mezi programy. Řešení tohoto problému vzešlo od Douga McIlroye a během jedné noci byl celý systém přepsán aby podporoval tzv. roury, které jsou jedou z důležitých součástí filosofie Unixu.

Po těchto krocích už byl Unix v podstatě vyzrálý operační systém. Šířil se mezi univerzitami i komerčními společnostmi a postupně se vylepšoval až nastal zlomový bod – verze 6. Vývoj Unixu byl rozdělen na dva hlavní proudy. Jedním z nich byl BSD [5] Unix byl vyvíjen na univerzitě Berkley v Kalifornii, z něhož byl v pozdější části vývoje (verze 4.1) vytvořen SunOS od společnosti Sun Microsystem, dnes známý jako Solaris. Z další verze BSD Unixu (4.2) pak byl vytvořen mikrokernelový operační systém Mach, ze kterého pak v konečném důsledku vznikl i Mac OS X. Je zřejmé, že toto období bylo pro vývoj Unixu velmi bouřlivé, a vznikalo mnoho verzí a odnoží původního systému, ale v podstatě dále zůstávaly mezi sebou více či méně kompatibilní.

Druhou odnoží se stal systém vyvíjený skupinou USL [6], z něhož pak vznikly systémy jako UnixWare nebo IBM AIX [7]. Z této odnože také vzniká pro budoucí Linux velmi důležitý Minix. Postupem času jsou do systémů přebírány vlastnosti z obou vývojových proudů a tak rozdíly mezi oběma odnožemi zanikají.

Kdy se tedy konečně dostaneme k Linuxu? V 80. letech založil Richard M. Stallman organizaci FSF (Free Software Foundation), jejímž cílem bylo vytvořit volně šiřitelné programy pro Unix, které by se pak daly použít i v komerčním Unixu. Tehdejším problémem ale bylo, že komerční Unixy byly velmi drahé, a nekomerční v

[5] Berkeley Software Distribution – odvozenina Unixu distribuovaná univerzitou Berkley v Kalifornii.

[6] Unix System Laboratories – Americká softwarová společnost.

[7] Advanced Interactive eXecutive – propertiální unixový operační systém firmy IBM.

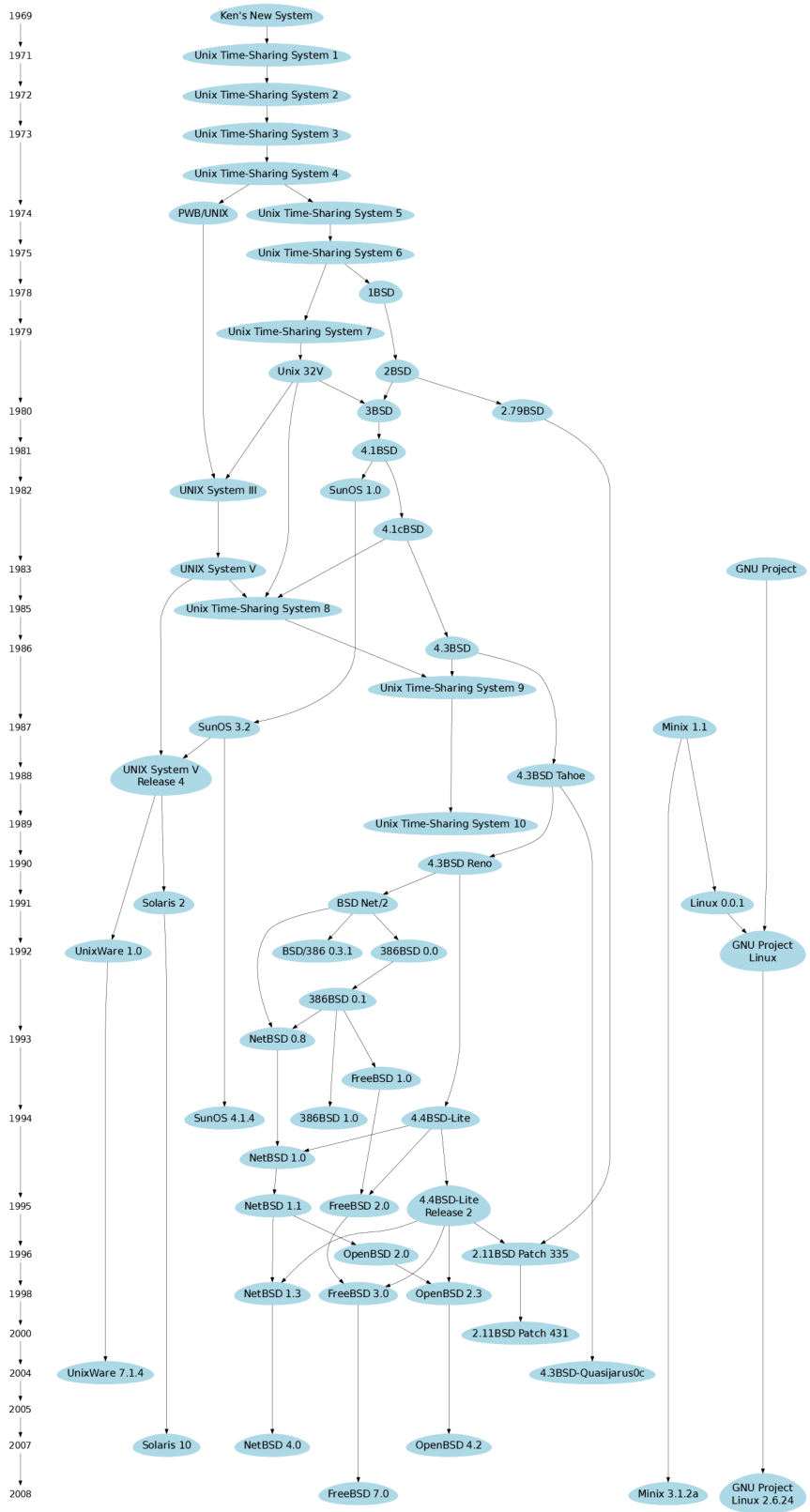
podstatě neexistovaly do chvíle, než právě Stallman v roce 1992 začal vyvíjet vlastní klon Unixu – GNU/HURD [8]. Vývoj tohoto systému se ale na velmi dlouhou dobu zastavil a i dnes velmi stagnuje.

To byla ale šance pro Linuse Torvaldse a jeho operační systém založený původně na Minixu prof. Tanenbauma. Linus postupně vyvíjel své jádro, a zdrojové kódy vystavoval veřejně na internetu, tak získal širokou podporu, a schopní lidé z celého světa začali do jeho systému přispívat. Tak byl v roce 1993 vytvořen plně použitelný, bezpečný a stabilní systém, který mohl klidně soupeřit s mnohými komerčními Unixy. Jeho vývoj probíhá dodnes v podstatě stejným způsobem pod vedením Linuse Torvaldse, za přispívání lidí a firem z celého světa.

Přibližně ve stejném roce (1993) vzniká také systém FreeBSD, jakožto pokračovatel systému BSD. Samozřejmě také pokračuje vývoj staršího systému NetBSD, který je v podstatě pokračovatelem BSD v.4.3. V roce 1995 pak z tohoto systému vzniká OpenBSD, jehož hlavní orientací je bezpečnost.

Následující obrázek Obr.2.1 ukazuje vývoj Unixu a jeho členění do nových větví od roku 1969 do roku 2008. Zobrazení není úplně kompletní, jelikož úplný graf se všemi větvemi, byť bezvýznamnými, by se zde ani zdaleka nevešel. Avšak myslím, že tento obrázek je pro ilustraci vývoje Unixu v tomto případě naprosto dostačující. Zdrojová data byla převzata z http://9grid.fr/articles/unix_diagram_simple.dot a graf byl vygenerován pomocí programu Graphviz.

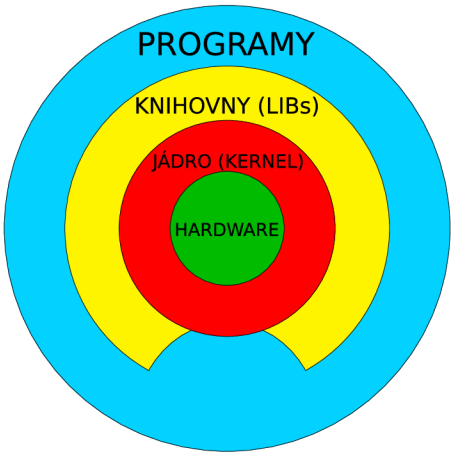
[8] HURD znamená Hird of Unix-Replacing Daemons a Hird znamená Hurd of Interfaces Representing Depth – je to v podstatě kolekce serverů běžícíhc na mikrokernelu, implementujících jednotlivé části OS. Náhrada Unixového kernelu.



Obrázek 2.1: Částečně zjednodušené zobrazení vývoje unixových systémů.

2.1.2 Filosofie

Jedním ze základních znaků operačních systému unixového typu je abstrakce hardwaru, a jeho odstínění od uživatelů a programů (viz. Obr.2.2). Jádro systému (kernel) je jakýmsi srdcem, které obstarává nízkoúrovňové operace nad hardwarem, a poskytuje uživatelům a programům standardní rozhraní pro práci s ním. V jádře jsou zakompilovány moduly (ovladače) pro práci s hardwarem, na kterém operační systém běží. Množství ovladačů, obecně modulů, zakompilovaných do jádra, by mělo být co nejmenší, resp. měly by zde být pouze opravdu ty nejnutnější funkce. A to z toho důvodu, že jakákoli chyba v jádře může mít za následek jeho zhroucení (kernel panic), resp. jakákoli chyba mimo jádro by neměla ohrozit běh systému.



Obrázek 2.2: Vrstvy unixového operačního systému.

Dalším důležitým rysem unixových systémů je připojování filesystému, který není členěn podle disků (A: B: C:), jak to známe např. z Dosu nebo Windows, ale disky jsou připojovány do jediného adresářového stromu, kde nejvýše umístěný adresář se nazývá root, a je to první část, která se namountuje při startu systému. Další disky (parititons) se pak připojují podle nastavení systému k jednotlivým adresářům ve stromové struktuře. Typicky pak na první pohled nepoznáte, kde začíná jeden disk (partition), a kde je již připojen disk jiný. Disky se tváří jako adresáře. To má samozřejmě své výhody.

V unixových systémech existuje pravidlo, že každý proces musí mít svého předka, resp. každý proces musí být spuštěn nějakým původním procesem. Tím prapůvodním procesem, předkem všech dalších procesů, je proces Init. Jeho PID [9] je 1, a již není součástí jádra. Další procesy se pak dají spustit dvěma způsoby, klonováním (fork) a samotným spuštěním (exec). Ke klonování procesu typicky dojde v případě, že nějaký program (proces) hodlá spustit jiný program. Původní program (proces) se

[9] Process identifier – identifikační číslo procesu.

naklonuje, spustí externí program, a sám sebe zničí. Nedojde tak ke změně dat a kódu původního procesu, a volaný program bude vědět komu vrátit výsledky.

Práce s pamětí je v operačních systémech velmi důležitá. V unixových systémech se používá tzv. stránkování. Znamená to, že pokud v systému běží dva stejné procesy, typicky od dvou různých uživatelů, pak nejsou v paměti nakopírovány dvakrát, ale část, která je nemodifikovaná, je zde pouze jednou, a sdílí se mezi jednotlivými instancemi programu. Pokud dojde ke změně, je uložena zvlášť. Je tedy zajištěno, že pokud dojde k situaci, kdy dva uživatelé pozmění data svého programu jiným způsobem, nedojde ke ztrátě těchto dat.

Unix byl od začátku vyvíjen jako multiuživatelský, multitaskový systém. To se projevilo také v jeho konstrukci a přísném oddělení jednotlivých uživatelských prostorů. Typicky to znamená, že jeden uživatel nemá právo zasahovat do procesu jiného uživatele, alespoň pokud na to nemá práva. V unixových systémech hrála odjakživa práva obrovskou roli. Základní rozdělení vypadalo tak, že existoval jakýsi *všemocný* uživatel *root*, typicky administrátor systému, a pak jednotliví uživatelé, jimž se mohla práva (čtení, zápisu a spouštění) libovolně nastavit. V dnešní době se zdá, že toto hrubé členění, root a uživatelé, přestává být dostačující, a proto vznikají projekty jako SELinux, nebo AppArmor, které umožňují jemnější nastavení práv a to i u superuživatele root. Odpadá nám tímto existence onoho *všemocného*, a systém se stává zase o něco bezpečnějším.

Roury jsou jedním z typických znaků unixových systému, které žádný uživatel jistě nepřehlédne, neboť je to něco, což přináší obrovské ulehčení práce. Důvod existence rour jsem již naznačil v předchozí kapitole. Jsou zde proto, aby se daly jednoduše předávat data mezi programy. Funguje to následovně. Jeden program čte data, typicky ze standardního vstupu, v našem případě třeba klávesnice, a vypisuje data na standardní výstup, v našem případě třeba monitor. Ale představte si případ, kdy za tento program rourou připojíme další program. První program v dobré víře, že zapisuje výstupní data na standardní výstup, je bude vlastně předávat na standardní vstup druhého programu, a ten, opět v dobré víře, že přebírá data ze standardního vstupu, je bude číst z výstupu programu předcházejícího. Jednoduše geniální.

A tímto se dostávám k *základní* filosofii unixových operačních systémů, a sice mít jednoduché, jednoúčelové programy, vykonávající právě tu jednu činnost, (zato velmi dobře) a kýženého cíle dosáhnout jejich vzájemným propojováním. Nejenže se pomocí tohoto přístupu minimalizují chyby v programech (jednodušší program zákonitě znamená menší počet chyb), ale také se nám naskýtá obrovská volnost v způsobech řešení problémů.

2.2 Operační systém GNU/Linux Fedora

Operační systém GNU/Linux Fedora (dále jen Fedora) je distribucí firmy RedHat, která se v prvé řadě zabývá vývojem komerční linuxové distribuce RedHat Enterprise Linux. Mimoto je také sponzorem komunitního projektu Fedora, jehož cílem je vytvoření svobodného operačního systému pro všeobecné použití. Fedora má kořeny v distribuci RedHat, kterou stejnojmenná firma vyvíjela do roku 2003, kdy byl jeho vývoj ukončen, a firma RedHat se zaměřila na komerční distribuci RHEL [10]. Ve stejné době vznikl komunitní projekt Fedora, který do verze 7 vyvíjel operační systém pod jménem Fedora Core. Od této verze byl název zkrácen na Fedora.

Fedora, stejně jako RHEL, je distribuce používající formát balíčků RPM. Abych byl přesnější, tento typ balíčků byl poprvé použit právě v RedHat Linuxu. Je to systém postavený na balíčkovacím systému Yum, který byl původně přejat z Yellow Dog linuxu a který je dále vyvíjen firmou RedHat. Fedora je distribuce známá především svou pokrokovostí, jednoduše řečeno, umožňuje to vývojářům RHEL vyzkoušet některé technologie a novinky, které by případně mohly být zahrnuty právě do této distribuce.

Fedora nabízí velké množství balíčků resp. programů, které z ní dělají všestranně použitelný systém. Může být tedy nasazena jednak jako desktopové prostředí, jednak jako serverové řešení. Je to systém, který poskytuje zejména vývojářům dobrou základnu a kvalitní rozhraní postavené na grafickém prostředí Gnome.

Odlišností Fedory, oproti ostatním distribucím, by se našlo hned několik. Zejména je to její balíčkovací systém Yum, ale také zaměření na bezpečnost. Jako jedna z prvních distribucí obsahuje v základní instalaci SELinux, jakožto nástroj pro detailnější definici práv jak uživatelů, tak superuživatele root. Není se však čemu divit, jelikož firma RedHat je velkým zastáncem tohoto řešení. Dalším takovým programovým vybavením, které je ve Fedoře velmi dobře podporováno je vývojové prostředí Eclipse, na kterém firma RedHat intenzivně pracuje a je také jejím přičiněním velmi dobře použitelné.

2.3 Databáze

Vzhledem k tomu, že při práci na tomto projektu jsem se musel potýkat s databázemi, nebylo by od věci přiblížit si tento systém trošku blíže. V první části této kapitoly se budu zabývat teoretickým základem databází a v druhé části konkrétním databázovým systémem MySQL, jelikož právě tento jsem v mé práci použil.

[10] RedHat Enterprise Linux – komerční sitribubce firmy RedHat.

2.3.1 Teorie databází

Databáze, tedy báze dat, je obecně soubor *nějakých* dat, které *někde* skladujeme. V mém případě je právě to *někde* paměťové médium, přesněji disk. Jde tedy o databázi elektronickou. Samozřejmě existuje několik typů databázových modelů. Já se ale budu zabývat výhradně databází relační. Ačkoliv panuje všeobecné přesvědčení, že relační model je relační díky vztahům mezi daty, není tomu tak.

Základním stavebním kamenem relačního modelu je relace, chcete-li množina dat, která je vybavena pomocnou strukturou. Tato pomocná struktura obsahuje jména atributů a popisy jednotlivých možných hodnot. Jednoduše řečeno lze relační databázi chápat, a také tomu tak většinou bývá, jako tabulku se sloupci, jako jména a popisy atributů, a řádky. V relačním modelu lze pracovat s tabulkou stejně jako s výsledkem dotazu.

Jelikož relace (tabulka) v databázi je uspořádána pouze do sloupců a řádků, a neexistuje první ani poslední řádek, nelze se přesně odkazovat na konkrétní řádek, musí struktura obsahovat nějakou konstrukci, která nám umožní přímo adresovat jednotlivé řádky. Této struktuře se říká *primární klíč*. Pomoci primárního klíče, který může být tvořen jedním nebo více atributy, můžeme jednoznačně identifikovat jednotlivé řádky v relaci.

Databázový systém samotný je tvořen v podstatě ze dvou částí. První jsou samotná data, a druhou jsou nástroje pro práci s těmito daty. Tyto nástroje většinou zajišťují ukládání, extrakci dat podle stanovených pravidel. V dnešní době se pro popis těchto pravidel používá téměř výhradně jazyk SQL [11], který byl vytvořen právě za tímto účelem, a poskytuje nám univerzální jazyk, jakým se můžeme s databází dorozumívat.

2.3.2 Databázový systém MySQL

MySQL je databázový systém, vyvíjeny švédskou firmou MySQL AB, a je vydáván po dvojí licencí. V mém případě budu používat MySQL pod licencí GPL. Tento systém, jak je patrno již z názvu, používá relační model databáze, a komunikace s ní probíhá prostřednictvím jazyka SQL. Je velmi oblíbený především ve spojení s webovým serverem Apache a skriptovacím jazykem PHP. Z uvedeného je tedy vidět, že se povětšinou používá pro tvorbu internetových stránek a aplikací. Je také obecně známo, že jde o velmi rychlou databázi, která ale svoji rychlost vykoupila menší škálou funkcí, takže záležitosti jako triggery nebo pohledy byly implementovány teprve relativně nedávno. Dá se ovšem říci, že je to vyspělá a velmi kvalitní databáze,

[11] Structured Query Language – strukturovaný dotazovací jazyk pro práci s daty v relačních databázích.

která má velmi dobré uplatnění, a nijak nezaostává za svými konkurenčními i open-sourcovými kolegy. V mém případě jsem použil MySQL ve verzi 5.0, což je v současné době nejvyšší stabilní verze.

MySQL se v širším pohledu skládá ze dvou základních částí. Ze serveru a klienta. Server jsem nainstaloval na můj notebook, budu tedy pracovat ve vlastním prostředí, bez nějakých větších či zásadnějších úprav. Součástí MySQL je také GUI [12] rozhraní MySQL Administrator, který ale v mém případě budu přehlížet, a veškerou práci s databází dubu obstarávat z příkazové řádky.

Jak jsem se již zmínil, databázový systém MySQL se ovládá pomocí jazyka SQL. Je tedy namístě zmínit se o několika základních příkazech. Princip tohoto databázového systému je jednoduchý, jak ukazuje Obr.2.3. Může tedy existovat několik nezávislých databází, a v nich několik tabulek. Samozřejmostí je možnost nastavení práv pro jednotlivé uživatele, ale tímto bych se v této práci nechtěl zabývat. Do databáze se tedy můžeme připojit následujícím příkazem, za předpokladu, že na lokální smyčce běží `mysqld` server.

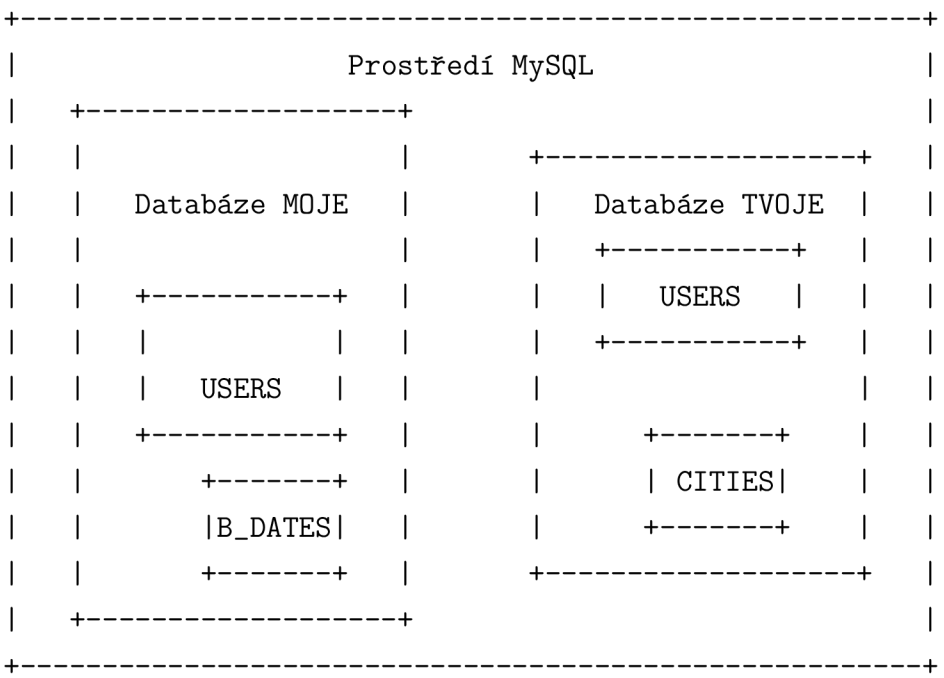
```
mysql -u <uživatel> -p -A <databáze>
```

Kde `<uživatel>` je jméno uživatele a `<databáze>` je jméno databáze, ke které se budeme připojovat. Tento parametr je však nepovinný a databázi si můžeme vybrat později. Přepínač `-p` pak říká klientovi, že pro uživatele je vyžadováno heslo.

Po připojení k databázi nás uvítá klient s hlášením o verzi a informacemi o nápovědě. Pokud jsme při spouštění neuvedli parametr pro výběr databáze, musíme si ji vybrat nyní příkazem `use <jméno databáze>` a můžeme začít pracovat. Jelikož jsem slíbil představit pár příkazů SQL, začnu tvorbou jednoduchých tabulek, na kterých budu dále prezentovat základní práci s tímto databázovým systémem. Půjde o tabulky, které jsem použil také ve své práci a sice tabulku IP adres a tabulku pro data z monitoringu. Relace mezi těmito dvěmi tabulkami bude 1:M.

Na tomto místě by jistě stál za zmínku i fakt, že samotný databázový systém MySQl podporuje několik druhů datových úložišť. Asi nejzajímavějšími z nich jsou *MyISAM* a *InnoDB*. Samozřejmě existují i další jako například BerkleyDB, Memory, CSV atd., ale jde o úložiště, která mají více specifická využití, a proto se o nich nebudu zmiňovat. Hlavním rozdílem mezi MyISAM a InnoDB je samozřejmě funkcionalita. Zatímco MyISAM se snaží být jednoduché, lehce použitelné, nenáročné a hlavně rychlé úložiště, které ale podporuje lepší indexování na sloupcích typu TEXT a BLOB a umožňuje oddělené uložení souborů s indexy a daty (např. na různé disky), je InnoDB zaměřeno na přidanou hodnotu a lepší funkcionalitu, ovšem za cenu rychlosti operací. InnoDB je tedy vybaveno funkcemi jako je možnost zamykání

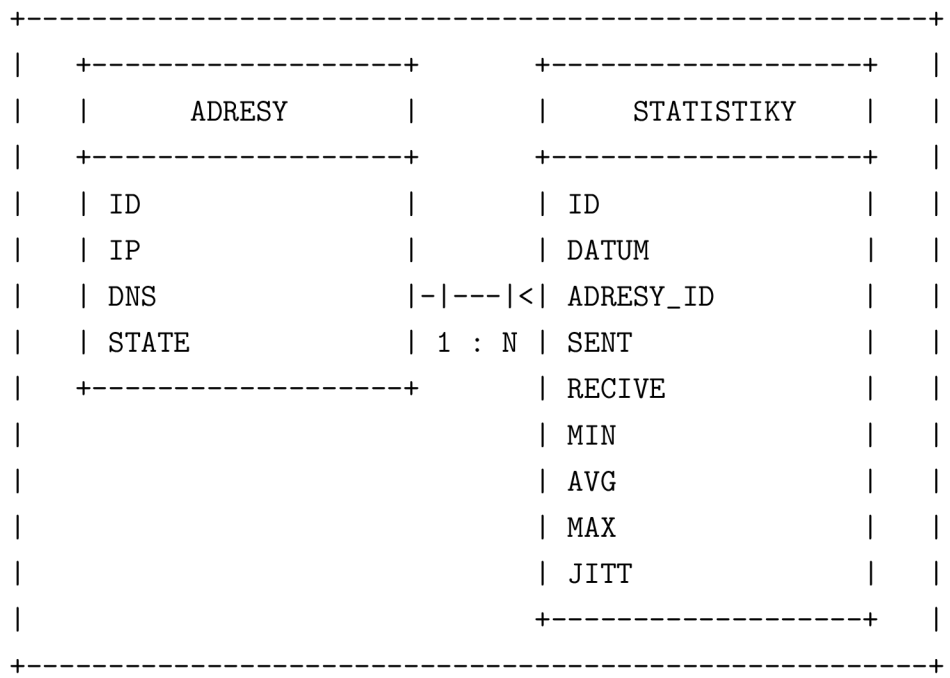
[12] Graphical User Interface – grafické uživatelské rozgraní



Obrázek 2.3: Jednoduchá ulázka struktury databází v MySQL.

jednotlivých řádků tabulek, dovede dokonce překročit maximální velikost souborů danou omezením filesystému a především umožňuje použití cizích klíčů. Zajišťuje také integritu dat a podporuje transakce. Já jsem ale pro potřeby tohoto projektu použil úložiště MyISAM, jelikož žádnou z přidaných funkcí nabízených úložištěm InnoDB nepotřebuji.

Tak tedy k vlastním tabulkám. Jak jsem již předeslal, půjde o dvě tabulky v relaci 1:M, což je vidět na Obr.2.4. Zatím se nebudu zabývat datovými typy sloupců a důvody, proč jsou takové, jaké jsou. Vše osvětlím až v praktické části. Takovéto dvě tabulky lze vytvořit jednoduchým SQL skriptem pomocí konstruktoru `CREATE TABLE`.



Obrázek 2.4: Relace mezi tabulkami v MySQL.

```
create table ADRESY (
ID          int unsigned primary key auto_increment not null,
IP          varchar(40) not null,
DNS         varchar(255),
STATE       tinyint default 1
) ENGINE = MYISAM;

create table STATISTIKY (
ID          int unsigned primary key auto_increment not null,
DATUM       datetime not null,
ADRESY\_ID  int unsigned not null,
SENT        tinyint,
RECIVE      tinyint,
MIN         float,
AVG         float,
MAX         float,
JITT        float
) ENGINE = MYISAM;
```

Tímto jednoduchým skriptem jsem vytvořil dvě tabulky. Skript ovšem nic neříká o závislostech mezi tabulkami. Datové úložiště MyISAM něco takového nehlídá, a tudíž je na nás, abychom se o vazbu postarali. To v praxi znamená, že jak při vkládání, tak při dotazování na data v tabulkách musíme mít namysli, že mezi tabulkami existuje určitá vazba. To většinou nebývá problém, protože přesně víme, jakou vazbu jsme mezi tabulkami chtěli udělat, a co má vazba vyjadřovat.

Našim tabulkám ovšem chybí ještě jedna podstatná věc. A sice indexy. Na položkách označených jako primární klíče se samozřejmě indexy vytvářejí automaticky. Ale co ostatní položky? V případě, že předpokládáme v tabulce statistik velké množství záznamů, řádově milióny, rozhodně by se nějaký ten index hodil, jelikož bez něj by mohly být operace nad tabulkou, přesněji dotazování na data, dosti pomalé. Je třeba ještě podotknout, že indexy sice zrychlují vytahování dat z databáze, ovšem také zabírají místo, a aby toho nebylo málo, negativně se projevují na rychlosti ukládání dat. Na tom ovšem není nic divného, když si uvědomíme, že při vložení každého záznamu se onen záznam musí na patřičných polích naindexovat, což logicky stojí určitý čas.

Indexy se na tabulkách samozřejmě dají přidělit již při vytváření tabulek, ale pokud máme již k dispozici data, která do tabulky chceme vložit, a těchto dat není zrovna málo, hodí se počkat s vytvořením indexů až po nahrnutí dat do databáze. Podstatně si tím zkrátíme dobu čekání, než se nám kýžená data dostanou do databáze. Samozřejmě si pak déle počkáme na vytvoření indexu, ale každopádně tím získáme. Jelikož jsou tabulky již vytvořeny, musíme použít příkaz pro změnu tabulky `alter table`. Tímto příkazem se dají samozřejmě provádět i další akce, jako například přidávání sloupců, jejích mazání, nebo změna datového typu sloupce. V takovém případě ovšem musíme být opatrní, abychom nepřišli o těžce vydobytá data.

```
alter table ADRESY add index (IP);
alter table STATISTIKY add index (DATUM);
alter table STATISTIKY add index (ADRESY_ID);
alter table STATISTIKY add index (AVG);
```

Touto jednoduchou úpravou jsme tabulky značně zrychlili. Abych pouze neuváděl domněnky, provedl jsem měření právě na těchto dvou tabulkách před a po naindexování, přičemž tabulka ADRESY obsahovala 709 záznamů a tabulka STATISTIKY plných 2 121 935 záznamů. Mezi jednotlivými měřeními jsem samozřejmě databázový server restartoval. Měření probíhalo na následujícím dotazu.

```
select * from STATISTIKY left join ADRESY on
(ADRESY.ID = STATISTIKY.ADRESY_ID) where ADRESY.IP="206.12.16.134"
and DATUM between "2008-02-25 17:50:00" and "2008-02-26 17:50:00";
```

V tabulce Tab.2.1 jsou vidět výsledky měření. Je tedy zřejmé, že indexy opravdu velmi urychlují práci s daty v tabulce. Ovšem i při naindexované tabulce hrozí nebezpečí, že indexy nebudou vůbec použity. Nastává to typicky v případech, kdy potřebujeme například spočítat nějakou hodnotu ze všech záznamů v tabulce. Existuje ale horší případ, a to je špatně napsaný dotaz. Totiž v případě, že dotaz formulujeme velmi nevhodně, můžeme tím zabránit různým optimalizacím, kterými může MySQL optimalizovat průchod databází, a v nejhorším případě se může stát, že bude nucen úplně ignorovat veškeré indexy a tabulku otrocky procházet celou, záznam po záznamu. Abychom však takovýmto případům předešli, existuje pomůcka ve formě příkazu `explain`, který vypíše způsob, jakým databáze data prochází a jaké indexy přitom používá a zda vůbec.

Tabulka	**Doba trvání**
Bez indexů	0,93 s
S indexy	0,27 s
Zrychlení	3,45 x

Tabulka 2.1: Výsledky měření vlivu indexů na rychlost databáze

Nyní ukážu velmi názorný příklad, jak se dá relativně velmi jednoduše napsat dotaz, který vlouděním malého nedopatření způsobí, že budou ignorovány některé indexy. Což může mít v důsledku celkem katastrofální dopad. Použijeme poněkud upravenou verzi dotazu předchozího, který ale bude z databáze vytahovat tatáž data. Za příkazem následuje výpis získaný pomocí explain.

```
select * from STATISTIKY WHERE
ADRESY_ID=(select ID from ADRESY where IP="206.12.16.134")
AND STATISTIKY.DATUM BETWEEN
"2008-02-25 17:50:00" and "2008-02-26 17:50:00";
```

```
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: STATISTIKY
         type: ref
possible_keys: ADRESY_ID,DATUM
          key: ADRESY_ID
      key_len: 2
          ref: const
         rows: 3280
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: SUBQUERY
        table: ADRESY
         type: ref
possible_keys: IP
          key: IP
      key_len: 42
          ref:
         rows: 1
        Extra: Using where
2 rows in set (0.00 sec)
```

Je vidět, že ikdyž dotaz není zrovna ideálně zapsaný, databáze se s ním vypořádala po svém a použila indexy, které se jí nabízely celkem bez problémů. Co když se ale vyskytne v dotazu nepříjemná chybička, která bude spočívat pouze v posunutí uzavření závorky až na konec řádku ?

```
select * from STATISTIKY  WHERE
ADRESY_ID=(select ID from ADRESY where IP="206.12.16.134"
AND STATISTIKY.DATUM BETWEEN
 "2008-02-25 17:50:00" and "2008-02-26 17:50:00");
```

```
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: STATISTIKY
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2121935
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: ADRESY
         type: ref
possible_keys: IP
          key: IP
      key_len: 42
          ref: const
         rows: 1
        Extra: Using where
2 rows in set (0.00 sec)
```

Na tomto příkladu je názorně vidět, jak jedna malá změna dotazu může způsobit katastrofu. Při procházení tabulky ADRESY sice bez problémů používá index, ale při prohledávání tabulky STATISTIKY nemůže index použít a je nucena ji procházet úplně celou, tedy 2 121 935 záznamů. Doba trvání takového dotazu byla na mém vlastní server 1 minutu a 16 sekund. Což je krajně nepřípustná doba.

Jako poslední ukázku možností psaní dotazu uvedu příklad, který z celkových statistik vybere ty ip adresy, které měly za dobu mezi 25.2.2008 17:50 a 26.2.2008 17:50 průměrnou dobu odezvy větší než 1000 ms a alespoň jeden z icmp [13] paketů se vrátil. Kromě těchto IP adres vypíše také celkový počet záznamů, které splňují podmínky pro danou IP adresu, průměrnou dobu odezvy a maximální dobu odezvy. Dotaz by vypadal asi takto, následuje také výsledek dotazu viz. Tab.2.2.

[13] Internet Control Message Protocol – internetový protokol pro přenos chybových a řídících zpráv mezi entitami v síti.

```
select IP,count(*) as pocet,avg(AVG) as prumer ,max(MAX) as maximum
from ADRESY left join STATISTIKY on (ADRESY.ID = STATISTIKY.ADRESY_ID)
WHERE RECIV <> 0 and DATUM BETWEEN
"2008-02-25 17:50:00" and "2008-02-26 17:50:00"
group by ADRESY.ID having prumer > 1000 order by pocet;

+-----------------+-------+-----------------+---------+
| IP              | pocet | prumer          | maximum |
+-----------------+-------+-----------------+---------+
| 158.130.6.254   |     9 | 1511.5518756443 | 4520.07 |
| 128.192.101.217 |    12 | 1854.8913491567 |  3563.7 |
| 193.55.112.40   |    21 | 2331.2175601778 | 4502.55 |
| 193.55.112.41   |    23 | 1766.2475488082 | 4377.13 |
| 204.56.0.138    |    24 | 1190.8269608816 | 3542.56 |
+-----------------+-------+-----------------+---------+
5 rows in set (0.19 sec)
```

Tabulka 2.2: Výsledek složitějšího SQL dotazu, který používa seskupování, aliasy a klauzuli having

A co dotaz vlastně udělal ? Pro všechny záznamy seskupené podle adres (`group by ADRESY.ID`) a splňující podmínku data a počtu přijatých paketů, spočítal počet celkových záznamů k dané adrese a průměrnou a maximální dobu odezvy. Vše nakonec porovnal s konstantou 1000 a vypsal pouze ty záznamy, které měly průměr větší než 1000 (v mém případě milisekund). Nakonec vše sestupně seřadil podle počtu záznamů. Je tedy vidět, že se dá napsat velmi jednoduše komplexní dotaz, který bude vytahovat z databáze právě ta data, která požadujeme.

Samozřejmě existuje ještě spousty dalších operací nad databází, které jsem zde neuvedl. Jsou to například triggery, pohledy, klauzule union a v neposlední řadě vkládání a aktualizace záznamů. Není mým cílem zde popsat veškerou funkčnost databázového systému MySQL, jen jsem chtěl nastínit způsob použití a ukázat některé jeho vlastnosti.

2.4 Skriptovací jakyk PHP

Skriptovací jazyk PHP [14] vznikl v roce 1994, kdy si pan Rasmus Lerdorf usmyslil vytvořit si systém počítání přístupů ke svým stránkám. Původně šlo jen o skript

[14] PHP: Hypertext Preprocesor – PHP: hypertextový preprocesor; Původní název Personal Home Page

napsaný v Perlu, ale postupem času byl přepsán do jazyka C, a spojil se s jiným jazykem téhož autora. Tak postupem času vznikal velmi šikovný skriptovací programovací jazyk, svou syntaxí velmi podobný jazyku C nebo Javě. Dnes existuje již ve stabilní verzi 5.2.6. Já budu ve své práci používat verzi 5.2.4.

PHP se hodí především k programování dynamických webových stránek a patří na rozdíl od Javy k serverovým skriptovacím nástrojům. Znamená to, že klientovi není na požádání posílán samotný PHP skript, ale již výsledek skriptu. Popravdě samotný skript by se ke klientovi ani dostat neměl, jelikož může obsahovat citlivé informace jako hesla k databázi apod. Používá se tedy k dynamickému vygenerování obsahu webové stránky, která se následně odešle, a webový klient nemá ani tušení, že byla stránka vygenerována nějakým skriptem.

Abych ovšem jazyku PHP nekřivdil, lze pomocí něj napsat téměř jakýkoli skript. Tedy nemusí nutně sloužit pouze ke generování webových stránek či webových aplikací. Je to v podstatě univerzální skriptovací jazyk s velmi dobrou podporou a rozsáhlými knihovními funkcemi. Co z něj tedy dělá jazyk předurčený ke generování webového obsahu? V prvé řadě je to zřejmě jeho původ, jelikož byl za tímto účelem vytvořen, dále také fakt, že velmi dobře pracuje s texty, snadno komunikuje s různými databázovými systémy, velmi si rozumí s webovým serverem Apache, je multiplatformní a v neposlední řadě je to projekt otevřený, tedy s rozsáhlou komunitou. Toto jsou důvody proč je PHP tak oblíbený mezi tvůrci webu. A právě pro časté spojení s webovým serverem Apache, databázovým systémem MySQL a operačním systémem Linux se zažila zkratka LAMP (Kombinace Linux + Apache + MySQL + PHP (Perl, Python) používaná pro webový server.).

Jak to tedy funguje ? Kdysi v dobách, když PHP ještě ani neexistovalo, vypadaly webové stránky tak jak byly napsány. Tzn. klient poslal požadavek na zobrazení webové stránky a server bez otálení odeslal stránku, která byla požadována, bez jakýchkoli úprav. Asi jako následující příklad.

```
<html>
Toto je obyčejná webová stránka.
</html>
```

To ale postupem času přestávalo stačit. Jazyk PHP tedy otevřel úplně nové možnosti tvorby webu. Lze díky němu psát rozsáhlé webové aplikace dokonce na takové úrovni, že již nejsou k rozeznání od aplikací jak je známe. Ovšem je nutno podotknout, že k takovým kouskům už samotné php nestačí a je třeba jej kombinovat například s javou.

Ale zpět k příkladu. Zatímco předchozí stránka se v prohlížeči zobrazí přesně tak jak je napsaná, stránka napsaná pomocí php je nejprve vyhodnocena, resp. je

proveden skript php, který stránka obsahuje a až následně je odeslána klientovi. Následuje příklad.

```
<html>
Toto je speciální špiónská stránka a zná vaši IP adresu.
Vaše IP adresa je  <?php echo $REMOTE_ADDR; ?>
</html>
```

Z uvedeného jsou zřejmé dvě věci. A sice, že kód php musí být uzavřen mezi značkami <?php a ?>, ale jsou přípustné i jiné jako <? a ?> nebo <% a %> a každý příkaz musí být oddělen středníkem. Na stránce, na které by se tedy nacházel výše uvedený kód, by se objevila vaše IP adresa. Způsob zápisu skriptu do stránky z výše uvedeného tak úplně nevyplývá. Lze jej totiž zapsat více způsoby.

- Vkládáním ostrůvků php kódu uzavřených mezi výše uvedenými značkami.
- Celý dokument může být php skript a html tagy lze zapisovat pomocí příkazu echo, resp. print.
- Soubor se může tvářit jako php skript, ale ve skutečnosti to může být čistě html.

Jak jsem již zmínil, php také velmi dobře pracuje s databázemi. V mém případě bude tou databází MySQL. Pomocí php se dá velmi jednoduše data do databáze ukládat, nebo je zase vytahovat. Ukážu jednoduchý příklad zadávání a vytažení dat z databáze pomocí php. Vkládat data budu pomocí následujícího skriptu. Je to mimochodem obdoba skriptu, který budu používat pro vložení stávajících dat do databáze.

```
<?php
  // připojení k databázi
mysql_connect("localhost", "jmeno", "heslo")\
or die ("Nelze otevrit databazi");
mysql_select_db("MOJE");
$cesta="/cesta/k/adresari/se/soubory";

if ($handle=opendir("$cesta")) {

    // prochází adresáře a hledá soubory
  while ($file=readdir($handle)) {
      // nesmí jít o hardlinky . ani ..
    if ($file != "." && $file != "..") {
        // otevření souboru pro čtení
      $fp = fOpen("${cesta}/$file","r") or die ("fOpen selhala");
```

```
      while (!(fEOF($fp))) {
          // načtení řádku ze souboru
        $radek= fGets($fp, 70);
        $prvky=explode(" ","$radek");
          // tvorba SQL dotazu
        @$sql="insert into STATISTIKY\
        (DATUM, IP, SENT, RECIV, MIN, AVG, MAX, JITT)\
        values ('$prvky[0]','$prvky[1]','$prvky[2]','$prvky[3]',\
        '$prvky[4]','$prvky[5]','$prvky[6]','$prvky[7]')";
          // zaslání dotazu do databáze
        mysql_query("$sql");
      }
    fClose($fp);
    }
  }
  closedir($handle);
  mysql_close;
}
?>
```

Stručně vysvětlím některé části skriptu. Nejprve je nutno navázat spojení a vybrat databázi, se kterou budeme pracovat. To vše je provedeno v prvních dvou řádcích. Vzhledem k tomu, že připojení k databázi s sebou přináší jistou režii, je lepší jej umístit mimo tělo cyklů. Příkaz `die()`, který se vykoná v případě, že `mysql_connect()` vrátí false, pouze vytiskne daný text a ukončí provádění skriptu. Slouží jako nejjednodušší ošetření chyb. Jako další následuje otevření adresáře pomocí `readdir()`. Jako parametr se této funkci předává cesta k danému adresáři. Následně bude skript načítat jednotlivé soubory v adresáři ovšem s výjimkou souborů . a .. , jelikož se jedná o hardlinky na stávající a nadřazený adresář. To je ošetřeno podmínkou `if`.

Vnořený cyklus pak prochází jednotlivé řádky daného souboru a pomocí funkce `fGets()` načítá data. Dalším krokem je rozdělení dat na části, které pak budeme ukládat do jednotlivých sloupců v databázi. To provedeme příkazem `explode()`, který daný řetězec rozdělí na jednotlivé položky do pole podle oddělovače. V tomto případě je tím oddělovačem mezera.

Dalším krokem je vytvoření SQL dotazu, což není nic až tak zajímavého, snad až na *zavináč* před příkazem. Ten je zde proto, že dovede potlačit téměř jakékoli chybové hlášení. Ve skriptu totiž předpokládáme osm položek na jednom řádku souboru, ale nemusí tomu tak být vždy. Může se totiž stát, že v případě když nedorazí žádná icmp odpověď, bude položek na řádku méně, a pak by došlo k chybě, která by narušovala běh skriptu. Jelikož nejde o nijak závažnou chybu, prostě jí pomocí @

skript ignoruje. Poslední zajímavou funkcí je pak `mysql_query()`, které předáme vytvořený SQL dotaz a funkce jej pak pošle ke zpracování databázi. Nakonec je ještě třeba pozavírat soubory a adresář, a ukončit spojení s databází.

Nyní přichází na řadu druhá část problému, a sice jak data z databáze dostat na webovou stránku. Řešení je pomocí skriptu php vcelku prosté. Data z databáze bude potřeba vytáhnout nějakým sql dotazem do proměnné. Pak je již stačí vypsat do stránky. Právě toto dělá následující skript. Jen podotknu, že do webové stránky vypíše data ve formě tabulky.

```
<html>
  <table>
    <tr>
      <th>IP</th>
      <th>DATUM</th>
      <th>SENT</th>
      <th>RECIV</th>
      <th>MIN</th>
      <th>MAX</th>
      <th>JITT</th>
    </tr>
    <?php
      mysql_connect("localhost", "jmeno", "heslo");
      mysql_select_db("MOJE");
      $sql="SELECT * FROM STATISTIKY";
      $vys= mysql_query("$sql");

      while ($zaznam=mysql_fetch_array($vys))
      {
        echo "<tr>";
        echo "<td>".$zaznam["IP"]."</td>";
        echo "<td>".$zaznam["DATUM"]."</td>";
        echo "<td>".$zaznam["SENT"]."</td>";
        echo "<td>".$zaznam["RECIV"]."</td>";
        echo "<td>".$zaznam["MIN"]."</td>";
        echo "<td>".$zaznam["MAX"]."</td>";
        echo "<td>".$zaznam["JITT"]."</td>";
        echo "</tr>";
      }
      mysql_close();
    ?>
  </table>
</html>
```

Tento skript je v zásadě velmi jednoduchý, pouze se připojí k databázi stejným způsobem jako v předchozím případě, sestaví SQL dotaz a pomocí funkce `mysql_query()`

jej pošle na databázový server. Výsledek této funkce můžeme uložit do proměnné a pomocí `mysql_fetch_array()` pak ve smyčce rozdělíme data na pole obsahující jednotlivé řádky tabulky. Indexy tohoto pole jsou pojmenovány jako odpovídající sloupce v tabulce. Není tedy problém data vytisknout do tabulky.

Možná ještě stojí za zmínku způsob, jakým php zachází s proměnnými. Php je slabě dynamicky typovaný jazyk, to znamená, že datový typ proměnné je spjat s její hodnotou, a tedy nepřiděluje se při deklaraci a proměnná může nabývat v podstatě hodnoty jakéhokoli typu. To, že je slabě typovaný navíc znamená, že není kontrolováno špatné použití datových typů a je ponecháno na programátorovi samotném, aby si to hlídal. Dále je třeba říci, že proměnná vzniká jejím použitím, (stejně jako její datový typ) a zaniká na konci funkce (pokud není statická), či programu, nebo použitím funkce `unset()`.

2.5 Webový server Apache

Apache je opensourcový webový server, který je pod správou komunity ASF [15] a vydávaný pod licencí AL [16], což je licence kompatibilní s GPL. Vznikl kolem roku 1993 ve společnosti NCSA [17] jako tehdejší Public Domain NCSA HTTPd. Hlavní vývojář ale roku 1994 opustil NCSA a celý vývoj se na nějakou dobu zastavil. Mezitím vývojáři a webamasteři z celého světa spolupracovali na vývoji patchů a opravách chyb tak dlouho, až byla ustanovena jakási komunita těchto lidí, kterým se podařilo v dubnu 1995 vydat první Apache 0.6.2. Ikdyž Apache vyžadoval ještě spousty práce a v podstatě bylo potřeba zásadní přepracování, stal se téměř okamžitě celosvětovým hitem. V prosinci roku 1995 pak bylo konečně vydána první stabilní verze Apache 1.0. V té době se již stal Apache nejpoužívanějším webovým serverem a tuto pozici neopustil dodnes.

V současné době je Apache ve stabilní verzi 2.2. To je také verze kterou budu používat já při své práci. Apache je ve svém základu velmi jednoduchý webový server avšak existuje pro něj nepřeberné množství modulů, které nabízejí pokročilejší a komplikovanější funkčnost. Moduly lze zapínat a vypínat virtuálním hostům po celém serveru. Ano, apache podporuje také virtuální hostování, to znamená, že na jednom serveru může běžet nezávisle na sobě více virtuálních webových serverů.

Jelikož je Apache velmi komplexní nástroj, pokud chceme opravdu využít jeho mimořádných schopností, je třeba trocha studia. Hlavním konfiguračním souborem, který by nás v takové případě velmi zajímal, je httpd.conf. Ani já jsem se nevyhnul

[15] The Apache Software Foundation

[16] Apache Licence – licence kompatibilní s GPL.

[17] Nationac Center fo Supercomputing Aplications – Národní středisko superpočítačových aplikací

malému zásahu do tohoto souboru, ale v podstatě, pokud instalujeme Apache z repozitářů té dané distribuce, měl by fungovat bez problémů již po nainstalování a spuštění deamona.

Jelikož správná konfigurace webového serveru Apache a jeho samotný konfigurační soubor httpd.conf, není zrovna triviální záležitost, a jde o informace, které by vyšly na knihu, nebudu se zde tímto zabývat, nehledě na to, že by to znamenalo další velmi podrobné studium.

3 PRAKTICKÉ ŘEŠENÍ PROJEKTU

Konečně se dostávám k samotnému řešení projektu, v této části postupně představím veškeré nástroje, které jsem vytvořil. Vysvětlím jejich funkci a popíšu způsob jejich použití. Tato kapitola bude tedy vhodná i jako jakýsi manuál, či howto, k použití těchto nástrojů.

Kapitola bude rozdělena na dvě části, v nichž v každé se budu věnovat jednomu z cílů této práce. Půjde v prvé řadě o skripty ke konfiguraci operačního systému Fedora a v druhé kapitole se budu věnovat samotnému monitoringu a zobrazování výsledků tohoto monitoringu.

3.1 Konfigurační skripty

Všechny skripty, které budu v této kapitole předvádět, jsou napsány pro příkazový interpret Bash a spolu dohromady splňují zadání projektu na vytvoření konfiguračních skriptů pro operační systém Fedora v síti Planetlab. Jde o v zásadě jednoduché skripty, jelikož konfigurace samotná není nijak složitý úkon, tedy za předpokladu, že víte co jak konfigurovat. Jde tedy především o to, že serverů v planetlabu je velmi velké množství, asi kolem 800.

Tyto nástroje, říkejme jim nástroj pro konfiguraci a nástroj pro kopírování, jsou postaveny na dvou základních programech. Jedním z nich je ssh a druhý je scp. Oba tyto programy používají velmi dobré šifrování a jsou tudíž velmi bezpečné. Šifrování je zajištěno protokolem SSH. Jen podotknu že v mém případě budu používat OpenSSH, což je opensource odnož od nyní již uzavřeného protokolu ssh. Pokud tedy budu mluvit o ssh, budu mít vždy namysli jeho otevřenou variantu.

3.1.1 Skript pro kopírování z resp. na servery

Tento skript již byl představen v mé semestrální práci a od té doby se nijak výrazně nezměnil, takže jen ve stručnosti připomenu jeho funkci a následně ukážu způsoby jeho použití. Jen připomenu, že tento nástroj používá jako stěžejní program, pomocí kterého se odehrává veškeré kopírování program scp.

Tento nástroj tedy slouží pro kopírování jednotlivých souborů, ale i celých adresářů na jeden či více serverů v planetlabu, nebo naopak, z těchto serverů v planetlabu může zkopírovat jednotlivé soubory, nebo celé adresářové struktury, zpět na počítač, ze kterého pracujeme. Skript vytváří také logovací soubor, do kterého jsou zapisovány úspěšně i neúspěšně provedené akce. V případě neúspěšných akcí se v logovacím souboru objeví také chybový výstup programu scp, nebo informace o tom, že cílový server nebyl v danou dobu dostupný. Dále pak skript vytváří také

soubor, do kterého ukládá adresy, na které nebo ze kterých, se nepodařilo provést kopírování. Je tedy možné to následně vyzkoušet znova pouze s adresami, které byly nedostupné.

Skript je také vybaven nápovědou, kterou je možno vyvolat spuštěním s parametrem -h, nebo při zadání nedostačujících či chybných údajů. V nápovědě je vcelku podrobně popsán způsob použití skriptu.

```
Skript pro upload souboru (adresaru) na nekolik serveru
pomoci programu scp.
Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>

Popis : Skript stahuje ci kopiruje soubory z nebo na servery,
    jejichz seznam se pripojuje jako druhy parametr,
    soubory pomoci programu scp. Pro stahovani ze serveru se
    pouziva prepinac -D. Pro Upload na servery se pouziva prepinac -U.
    V pripade stahovani souboru jsou v cilovem adresari vytvoreny
    adresare pojmenovane jako IP serveru, ze kterych se do techto
    adresaru soubory stahuji.
    Zdrojove a cilove adresare jsou v souboru ktery se pripojuje jako
    treti parametr.

Format seznamu souboru:
    [zdroj][mezera][cil][mezera][prepinac]

    Pozn. mezera muze byt nahrazena jakymkoli bilym znakem
    krome konce radku a cilova adresa nemusi existovat ale
    musi byt zadana jako adresar! Prepinac neni povinny
    parametr a pouziva se predevsim -r pro rekurzivni
    kopirovani adresaru.

Pouziti : "$0" [ -h ] [-D|-U file1 ] [ file2 ]
Popis parametru :
    [ -h | --help]  Vypise tuto napovedu a bezchybne skonci
    [-D]            Prepinac pro stahovani souboru ze serveru
    [-U]            Prepinac pro kopirovani souboru na servery
    [ file1 ]       Soubor obsahujici adresy serveru ze kterych se
                    ma stahovat
    [ file2 ]       Soubor obsahujici adresy souboru ktere se maji
                    stahovat a adresy kam se maji stahovane soubory
                    ukladat. Zdroj a cil musi byt na jednom radku
                    oddelen alespon jednou mezerou.
```

Jak je vidět, nápověda je celkem rozsáhlá a není problém z ní pochopit způsob použití. Ovšem může se stát, že dojde k situaci, kdy si uživatel nebude vědět rady i po přečtení nápovědy. V takových chvílích asi nezbývá nic jiného, než nahlédnout do

zdrojového kódu skriptu a na vlastní oči se přesvědčit jak vlastně funguje. Naštěstí je kód dobře okomentovaný, takže by pro nikoho neměl být problém se v něm dobře vyznat.

Nyní předvedu pár příkladů použití. Představme si situaci, kdy chceme na servery nakopírovat nějaká data. V prvé řadě to znamená vytvořit si soubor s adresami, na které budeme chtít tyto soubory nakopírovat. Dejme tomu, že si tento soubor pro názornost nazveme adresy.txt. Jeho obsah bude vypadat například následovně.

```
195.113.161.83
12.46.129.14
63.64.153.84
70.137.68.249
128.42.6.145
```

Použil jsem sice pouze 5 serverů, ale není problém použít i více, třeba i stovky serverů. Dalším souborem, který budeme potřebovat je seznam souborů které budeme chtít kopírovat. Podle nápovědy je zřejmé, že každý soubor musí být umístěn na jednom řádku, a musí být uveden zdroj, v našem případě cesta na lokálním stroji. Musí být uveden také cíl, v tomhle případě to bude cesta na vzdáleném stroji. Vzhledem k tomu, že na všech serverech v planetlabu na které se připojíme existuje adresář /home/cesnet_vutbr1/ není problém zadat i absolutní cestu. Budeme tedy chtít například kopírovat dva soubory a jeden adresář, který sám bude obsahovat další soubory. Seznam souborů tedy bude vypadat následovně a uložíme jej pro názornost jako soubory.txt.

```
/home/Moje/Skripty/scp/soubor1 /home/cesnet_vutbr1/
/home/Moje/Skripty/scp/soubor2 /home/cesnet_vutbr1/
/home/Moje/Skripty/scp/Adresar /home/cesnet_vutbr1/ -r
```

Pro lepší představu, jak vypadá adresářová struktura testovaných souborů přikládám výpis ls -1R.

```
$ ls -1R.:
Adresar
soubor1
soubor2

./Adresar:
soubor3
soubor4
```

Nyní už je vše připraveno pro kopírování. Předpokládám, že skript je umístěn v adresáři spolu se souborem s adresami a seznamem souborů ke kopírování. Následující výpis ukazuje spuštění a průběh kopírování.

```
$ sh scp_download.sh -U adresy.txt soubory.txt
Kopiruji soubory na cesnet_vutbr1@195.113.161.83 :
soubor1                              100%  261     0.3KB/s   00:00
soubor2                              100%  261     0.3KB/s   00:00
soubor3                              100%  261     0.3KB/s   00:00
soubor4                              100%  261     0.3KB/s   00:00
Kopirovani souboru na cesnet_vutbr1@195.113.161.83 probehlo v poradku
Kopiruji soubory na cesnet_vutbr1@12.46.129.14 :
soubor1                              100%  261     0.3KB/s   00:00
soubor2                              100%  261     0.3KB/s   00:00
soubor3                              100%  261     0.3KB/s   00:00
soubor4                              100%  261     0.3KB/s   00:00
Kopirovani souboru na cesnet_vutbr1@12.46.129.14 probehlo v poradku
Kopiruji soubory na cesnet_vutbr1@63.64.153.84 :
soubor1                              100%  261     0.3KB/s   00:00
soubor2                              100%  261     0.3KB/s   00:00
soubor3                              100%  261     0.3KB/s   00:00
soubor4                              100%  261     0.3KB/s   00:00
Kopirovani souboru na cesnet_vutbr1@63.64.153.84 probehlo v poradku
Kopiruji soubory na cesnet_vutbr1@70.137.68.249 :
soubor1                              100%  261     0.3KB/s   00:00
soubor2                              100%  261     0.3KB/s   00:00
soubor3                              100%  261     0.3KB/s   00:00
soubor4                              100%  261     0.3KB/s   00:00
Kopirovani souboru na cesnet_vutbr1@70.137.68.249 probehlo v poradku
Kopiruji soubory na cesnet_vutbr1@128.42.6.145 :
err5 : Server nedostupny : udalost ulozena do scp_download.sh.log
Na nekterych serverech probehlo kopirovani neuspesne
viz. unreachable.txt. Pocet neuspesnych kopirovani : 1
Konec skriptu..
```

Výstup programu je tedy dostatečně přehledný a v každém momentě víme, co se právě odehrává. Vidíme také, že nakonec dojde k jakési sumarizaci, kdy se vypíše počet neúspěšných kopírování a odkazem na logovací soubor. Aby byla ukázka funkčnosti tohoto nástroje kompletní, ještě doplním, že toto byla pouze ukázková zkouška, v reálném případě by bylo lepší zkopírovat celý adresář obsahující kopírované soubory, ale mým cílem bylo ukázat, že se dá vše opravdu rozepsat, což znamená volnost v možnostech použití. Následuje výpis logovacího souboru.

```
[20080507110410] Kopirovani souboru na
    cesnet_vutbr1@195.113.161.83  probehlo v poradku
[20080507110422] Kopirovani souboru na
    cesnet_vutbr1@12.46.129.14  probehlo v poradku
[20080507110435] Kopirovani souboru na
    cesnet_vutbr1@63.64.153.84  probehlo v poradku
[20080507110509] Kopirovani souboru na
```

```
    cesnet_vutbr1@70.137.68.249  probehlo v poradku
[20080507110511] err5 : Server nedostupny : 128.42.6.145
```

Myslím, že jsem dostatečně objasnil funkci tohoto nástroje, ale pro názornost ještě zopakuji postup použití a na Obr.3.1 pak můžete vidět znázornění jak toho jak nástroj funguje.

- Vytvořit soubor s IP adresami, nebo dománovými názvy jednotlivých uzlů v planetlabu.
- Vytvořit seznam souborů a jejich zdrojové a cílové cesty, popřípadě s přepínači
- Spustit skript s parametry -D adresy soubory.
- Zkontrolovat log soubor
- Případně opakovat skript, ale pouze s adresami, které byly nedostupné. Lze je nalézt v souboru unreachable.txt.

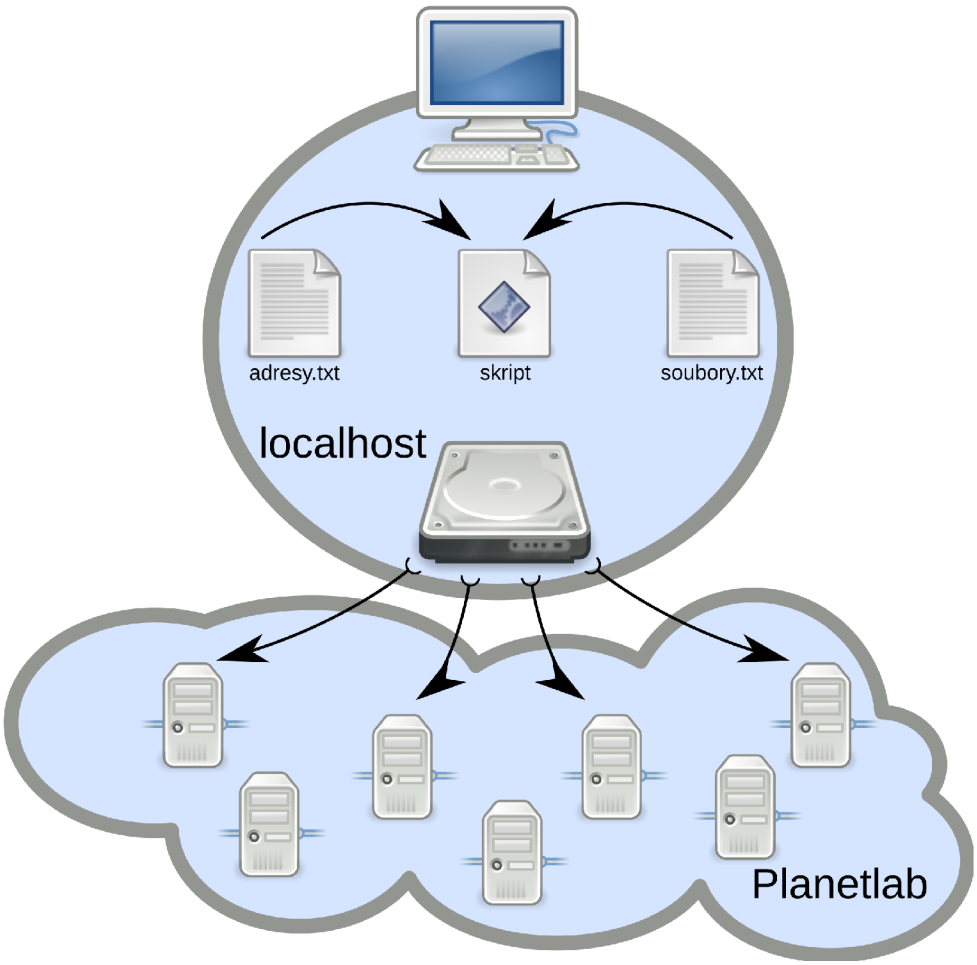
Další funkcí nástroje pro kopírování je přenos souborů z uzlů v planetlabu na svůj vlastní stroj. Probíhá to zcela analogicky ke kopírování na servery planetlabu, ovšem s jedinou malou výjimkou. Zdrojové a cílové cesty se v seznamu cest musí přehodit a na cílovém stroji se vytvoří v cílové cestě adresáře pojmenované po IP adresách jednotlivých serverů. Do těchto adresářů se budou ukládat vždy data od daného serveru.

Pro názornost opět předvedu, způsob jakým kopírování provést. Soubor s IP adresami cílových uzlů již máme přichystaný a v tomto testu tedy zůstává nezměněn. Musíme ale pozměnit seznam souborů, i v případě, že budu kopírovat právě ty soubory, které jsem v předchozím testu nakopíroval na servery. Seznam těchto souborů je třeba přehodit tak, aby první byla vždy cesta na zdrojovém stroji a druhá cesta na cílovém stroji. Pro tento případ bude seznam vypadat následovně.

```
/home/cesnet_vutbr1/soubor1 /home/Moje/Skripty/scp/zpet/
/home/cesnet_vutbr1/soubor2 /home/Moje/Skripty/scp/zpet/
/home/cesnet_vutbr1/Adresar /home/Moje/Skripty/scp/zpet/ -r
```

Nyní již zbývá pouze provést skript. Nebudu zde nyní ukazovat kompletní terminálový výstup z průběhu skriptu, jen vězte, že je velmi podobný předchozímu. Vlastně se na tom ani nemá moc co měnit. Skript ovšem musíme spustit s parametrem `-D`, jelikož se v tomto případě jedná o download dat.

```
sh scp_download.sh -D adresy.txt soubory.txt
```



Obrázek 3.1: Znázornění funkce nástroje pro kopírování.

V obrázku jsou některé části převzaty z projektu Gnome viz, www.gnome.org.

Přikládám ještě výpis adresářové struktury, které se vytvořila na mém stroji v adresáři `zpet/`, přesně tak, jak je zapráno v seznamu souborů. Adresář `zpet/` ovšem před spuštěním skriptu neexistoval. Skript tedy vytvoří adresářovou strukturu tak, jak je zapsána v seznamu souboru v případě, že některé cílové adresáře neexistují.

```
zpet/:
12.46.129.14
195.113.161.83
63.64.153.84
70.137.68.249

zpet/12.46.129.14:
Adresar
soubor1
soubor2
```

```
zpet/12.46.129.14/Adresar:
soubor3
soubor4
.
.
```

Výpis není kompletní, jelikož jsou obsahy všech adresářů stejné rozhodl jsem se výpis značně zkrátit. Obsah adresáře `zpet/` je ale zachován kompletní a je vidět, že v něm chybí jedna adresa. To je způsobeno tím, že stejně jako v předchozím příkladě, byl poslední server nedostupný a tím nedošlo ke kopírování dat ani v jednom směru.

3.1.2 Skript pro konfiguraci serverů

Konfigurace uzlů v planetlabu je velmi důležitou součástí projektu. Nástroj pro konfiguraci serverů, který v této části představím je nepostradatelný ve chvílích, kdy je potřeba na všech, nebo nějakém větším okruhu serverů provést nějakou zautomatizovanou činnost. Jelikož je v podstatě nemožné, aby byl každý server konfigurován zvlášť. Tím, že je to skript napsaný pro příkazový interpret Bash, se práce ještě zjednodušuje, jelikož na jednotlivých uzlech sítě Planetlab běží operační systém Fedora, který používá implicitně právě tento příkazový interpret.

Tento skript je postaven na standardním programu ssh, který je součástí snad všech linuxových distribucí. Umožňuje přístup ke vzdálenému shellu serveru, ke kterému se připojujeme. Navíc provádí pomocí protokolu SSH autentizaci a šifrování přenosu, je to tedy velmi bezpečný způsob, jak přistupovat ke vzdálenému stroji. Za normálních okolností, je možné se ke stroji připojit, provádět na něm nějakou činnost a následně se zase odpojit. V tomto případ se bude skript moci chovat dvěma způsoby. Jednak, jak jsem zmínil, připojit se, provést nějakou činnost a zase se odpojit, a jednak spustit činnost, program nebo skript na pozadí a odpojit se aniž by daná činnost skončila.

Skript samozřejmě obsahuje i vestavěnou nápovědu, která je, jako u předchozího skriptu celkem obsáhlá a neměl by být problém se podle ní řídit. V případě, že bude potřeba si skript upravit, či odvodit jeho funkci ze zdrojového kódu, taktéž by neměl být problém, jelikož skript je poměrně dobře okomentovaný. Nápověda tedy vypadá následovně.

```
Skript pro vzdalenou konfiguraci serverů.

Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>

Popis : Spusti zadany prikaz na vzdalenych strojich definovanych
        v seznamu zadavanem jako prvni parametr. Misto prikazu
        lze pouzit i skripty, ktere jsou ulozeny v ./scripts/.
        V pripade tvorby vlastnich skriptu nezapomente na konec
        vlozit "rm -f $0", aby se na disku nehromadily zbytecne
        soubory. Jakoukoli cinnost lze spustit i na pozadi,
        tak aby bezela i nadale po odpojeni od serveru.

Pouziti :
  ssh_skript.sh [ -h ] [ file1 ] [ -s ] [ -b ] [ command|script ]
Popis parametru :
        [ -h | --help]  Vypise tuto napovedu a bezchybne skonci
        [ file1 ]       Soubor obsahujici adresy cilovzch serveru
        [ -s ]          Parametr, ktery rika ze se ma pouzit
                        hotovy skript z adresare ./scripts/
        [ -b ]          Parametr spusti danou akci na vzdalenem
                        stroji na pozadi a uzavre standardni
                        vystup, vstup a chybovy vystup, aby akce
                        pokracovala i po odpojeni
        [ command|script ]
                        Muze byt jednotlivy prikaz s parametry,
                        nebo  i vice prikazu spojenych rourami,
                        ci oddelenych strednikem, nebo skript
                        ulozeny v ./scripts/, v tomto pripade
                        zadavat bez pripony a pouzit prepinac -s

Priklad :
     sh ssh_script.sh adresy.txt uname -a  -- vypise informace o
     kazdem cilovem stroji definovanem v souboru adresy.txt

     sh ssh_script.sh adresy.txt -s -b calculate -- spusti skript
     calculate z adresare ./scripts/ a necha jej bezet i po
     odpojeni

Poznamka : Opatrne pri spousteni skriptu na pozadi, za vsech
           okolnosti se bude hlasit spravne provedeny skript
           a to i v pripade fatalni chyby ve spoustene akci.
```

Jak je vidět, skript má dva volitelné parametry, které výrazným způsobem ovlivňují jeho chování. Jeden z nich, parametr `-b` způsobí, že skript či program poběží na vzdáleném stroji i po odpojení, a ssh nebude čekat na jeho ukončení. Druhý parametr `-s` způsobí, že bude spuštěn skript z adresáře ./scripts/. Tuhle funkci jsem imple-

mentoval hlavně z úvodu, že se může vyskynout, a dle mého názoru také velmi často vyskytne, potřeba opakovaně spouštět některé skripty. Takto je zajištěna lepší správa těchto skriptů i jejich bezproblémové použití.

Předvedu pár příkladů použití tohoto skriptu a následně popíšu, jak vlastně skript uvnitř funguje a detailně rozeberu některé jeho části. Nejdříve jednoduchý příklad. V minulé kapitole jsem na servery kopíroval nějaké soubory a adresářem, které tam již zůstaly. Není ovšem zrovna vhodné nechávat na serverech v planetlabu nepotřebné soubory a proto je právě pomocí tohoto skriptu vymažu. Příprava je jednoduchá, stačí soubor s adresami serverů, ze kterých budu chtít data vymazat, a ten zůstal nezměněn od minula. Dále potřebuji samozřejmě také znát soubory, které hodlám odstranit, což je pochopitelné. Ty také znám. Vše je tedy připraveno a mohu spustit skript.

```
$ sh ssh_execute.sh adresy.txt rm -rfv soubor1 soubor2 Adresar
Server: 195.113.161.83
removed ‘soubor1’
removed ‘soubor2’
removed ‘Adresar/soubor3’
removed ‘Adresar/soubor4’
removed directory: ‘Adresar’
Skript probehl v poradku
Server: 12.46.129.14
removed ‘soubor1’
removed ‘soubor2’
removed ‘Adresar/soubor3’
removed ‘Adresar/soubor4’
removed directory: ‘Adresar’
Skript probehl v poradku
Server: 63.64.153.84
removed ‘soubor1’
removed ‘soubor2’
removed ‘Adresar/soubor3’
removed ‘Adresar/soubor4’
removed directory: ‘Adresar’
Skript probehl v poradku
Server: 70.137.68.249
err5 : Server nedostupny : udalost ulozena do ssh_execute.sh.log
Server: 128.42.6.145
err5 : Server nedostupny : udalost ulozena do ssh_execute.sh.log
```

Použití tohohle skriptu je tedy vcelku jednoduché, stačí zadat soubor s adresami napsat příkaz s parametry a vše funguje jak má. Tímto způsobem se dá velmi jednoduše konfigurovat cílový server, ať už to znamená spouštění různých deamonů, nastavování parametrů, změny konfiguračních souborů. Ovšem v případě,

že chceme spustit nějaký dlouho běžící program či skript, nastává problém, že tento bude ukončen při odpojení od serveru. Je to způsobeno tím, že spuštěný program, ikdyž třeba na pozadí, je potomkem daného sezení (bashe) a tedy po ukončí tohoto sezení bude ukončen i tento potomek.

Řešení tohoto problému existuje několik. Prvním, které asi každého napadne, je zřejmě `nohup` pomocí něhož lze spustit příkaz tak, že je odolný proti signálu hangup. To ovšem způsobí, že ssh zůstane po odpojení "viset" a z nějakého důvodu čeká na dokončení skriptu. Další možností, která by se mohla zdát přijatelná je příkaz `setsid`, který spustí daný příkaz v novém sezení. Ovšem při odpojení od stávajícího sezení zůstane ssh opět "viset". Dle mého názoru, je t způsobeno tím, že nové sezení je také potomkem stávajícího, a tak se ssh nemůže odpojit, protože stále čeká na ukončení procesu svého potomka.

Asi nejlepší způsob jak dosáhnout toho, aby nějak program zůstal běžet, ikdyž se od stroje odhlásíme, je podle mého názoru použití programu `screen`. Přináší to i další výhody, jako možnost znovu získat výstup a vstup programu, který jsme takto opustili. Ovšem pro potřeby konfigurace, či spouštění úloh, nebo programů na jednotlivých uzlech planetlabu je to nevhodné a to nejen proto, že kromě programu musí na serveru zůstat běžet také screen, ale zejména proto, že na serverech v planetlabu není tento program přítomen. Nemluvě o tom, že nic tak komplexního není potřeba.

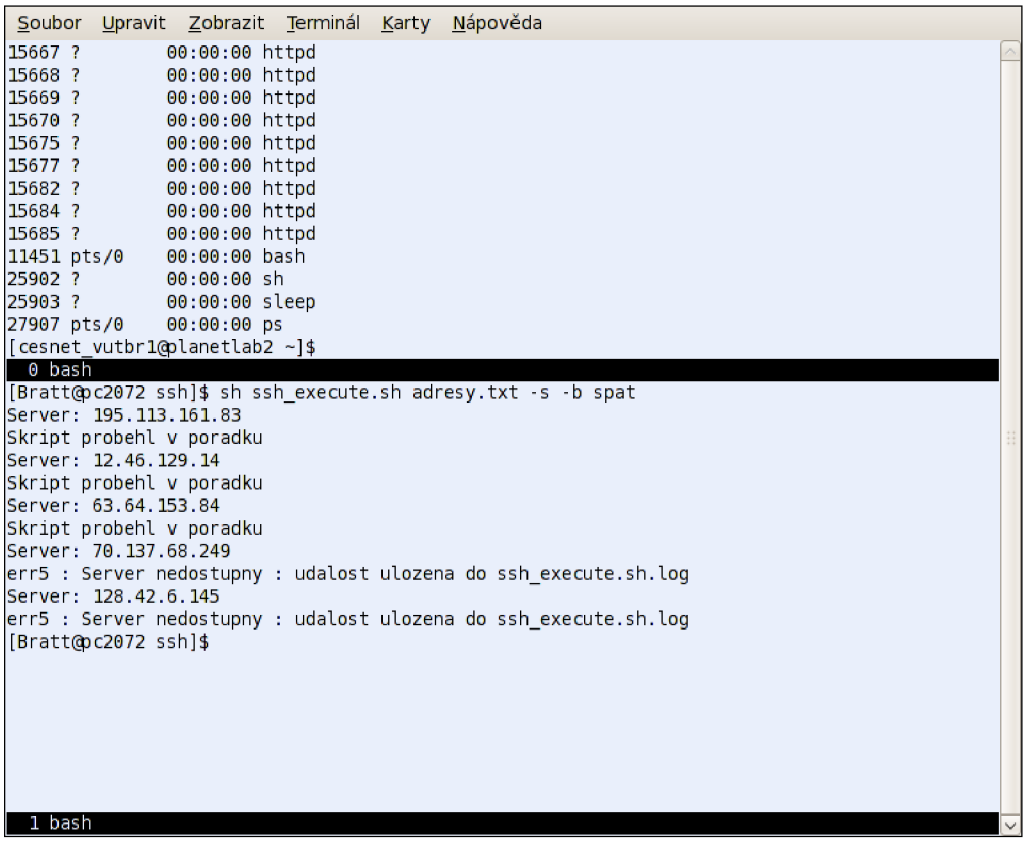
Dospěl jsem k názoru, že nejlepším způsobem jak docílit toho, aby procesy zůstaly běžet i po odhlášení je v tomto případě uzavření standardního vstupu, výstupu a chybového výstupu a poslat programu na pozadí. Nese to s sebou i jisté nevýhody. Jednou z nich je, že se nikdy nedovíme, zda nedošlo k chybě hned při spuštění skriptu, jelikož chybový výstup je uzavřen. Podobný problém nastává v případě, kdy chceme pomocí tohoto skriptu spustit nějaký příkaz a jeho výstup směrovat do souboru, či na jeho vstup směrovat nějaký soubor. V takovém případě totiž nebude přesměrování fungovat, jelikož standardní vstup i výstup jsou taktéž zavřeny. To ale nevidím jako zásadní problém, jelikož na takovéto operace je lepší použít externí skript, který pak funguje bez problémů tak, jak má, jelikož směrování budeme provádět uvnitř skriptu.

Předvedu tedy názorný příklad. Představme si, že chceme spustit na několika serverech skript, který máme připravený. V tomhle případě jde pouze o velmi jednoduchý skript, který má sloužit jenom jako ukázka toho, jak to funguje. Analogicky pak lze spustit jakýkoliv jiný skript či program.

```
#!/bin/bash
sleep 90
rm -f $0
```

Skript má pouze za úkol na daném počítači vykazovat jistou činnost. Tou činností je sleep po dobu 90 sekund. Dobu jsem si zvolil tak abych se stihl bez problémů připojit ke vzdálenému serveru a přesvědčit se, že skript opravdu běží. Skript pojmenuji spat.sh. Skript tedy spustíme následovně a na Obr.3.2 je zachycen výstup skriptu a výpis procesů na vzdáleném stroji, kde jde opravdu vidět, že sleep běží.

```
sh ssh_execute.sh adresy.txt -s -b spat
```



Obrázek 3.2: Výstup skriptu pro konfiguraci.

Je vidět, že skript funguje bez problémů a ve spojení s předchozím skriptem na kopírování souborů jsou možnosti v podstatě neomezené.

Co se týče vnitřní funkce skriptu, není to vlastně nic složitého. Pominu-li kontrolu parametrů a existence souboru s adresami, veškeré dění se odehrává v jediné smyčce. Nejdříve je třeba zkontrolovat, zda je daný server vůbec dostupný. Toto je vyřešeno jednoduchým pingem. Další věcí, která se ve smyčce rozhoduje co se provede, je přítomnost, či nepřítomnost parametru `-s`, který skriptu říká, že má použít externí skript. V případě, že je parametr `-s` přítomen, je externí skript pomocí scp zkopírován na cílovou stanici do domovského adresáře, (kam se mimochodem ssh i scp připojuje ve výchozím nastavení) a následně spuštěn pomocí ssh.

V případě, že je přítomen také parametr -b, je skript na cílovém stroji spuštěn s parametry `2<&- 1<&- 0<&- &`. Což znamená, pokud postupujeme zleva, zavření chybového výstupu, zavření standardního vstupu, zavření standardního výstupu a poslední ampersand to celé spustí na pozadí.

Pokud ale parametr `-s` přítomen není, znamená to, že bude proveden příkaz, tak jak byl uživatelem napsán. V případě nastaveného parametru -b jsou pak ještě přidány další parametry, stejné jako v předchozím případě.

Celá smyčka tedy ve výsledku vypadá takto.

```
for ip in $adresy; do
  echo "Server: $ip"

    # kontrola dostupnosti
  dostupnost $ip
  [ $? -ne 0 ] && failed 5 $ip && continue

    # Jako parametr byl predan skript
  if [ $skript -ne 0 ]; then

    launch="scripts/${1}.sh"

      # kontrola yda externi skript existuje
    [ -s $launch ] || print_err 4

      # Skript je nutno nakopirovat na vydaleny stroj
    scp -r $launch $username@$ip:.$$skript.sh &> /dev/null

      # Sousteni skriptu na vydalenem stroji
    ssh $username@$ip sh .$$skript.sh $bckg 2>> $log
    [ $? -eq 0 ] && success $ip || failed 6 $ip;

    # Jako parametr byl predan prikaz
  else
      # Provedeni prikazu na vzdalenem stroji
    ssh $username@$ip $@ $bckg 2>> $log
    [ $? -eq 0 ] && success $ip || failed 6 $ip;
  fi
done
```

Celý skript spolu s ostatními vytvořenými v tomto projektu je zařazen v příloze. Jen bych rád podotkl, že využití těchto dvou skriptů pro konfiguraci operačního systému Fedora (obecně však téměř jakéhokoli linuxového operačního systému) v síti Planetlab (i mimo ní), záleží pouze na uživatelích těchto skriptů. Mým cílem bylo pouze vytvořit nástroje a ty dobře okomentovat a přiložit nápovědu, aby se

uživatel vyznal v použití. Není však v mé moci zabránit uživatelům těchto skriptů v jejich chybném použití, ikdyž jsem se všemožně snažil, aby byl způsob použití co nejjednodušší.

3.2 Monitoring a statistiky

V této kapitole bych rád navázal na mou předchozí práci *Skripty pro instalaci a konfiguraci operačního systému Fedora*, kde jsem nastínil možnosti generování statistických dat pro sledování tzv. zdraví planetlabu. Základní myšlenka v podstatě zůstala stejná, ovšem hodně se toho muselo změnit. Můj původně zrealizovaný nápad, generovat pomocí skriptu statistická data o dostupnosti či nedostupnosti serverů v Planetlabu, a jejich ukládání do systému adresářů se brzy stal nedostačující. Bylo totiž celkem časově náročné z takto uložených dat získat nějaké komplexní informace o stavu Planetlabu či jednotlivých uzlů. Nástroje, které jsem za tímto účelem vytvořil byly vcelku pomalé a získání informace z těchto dat se tak poměrně komplikovalo. Přišel tak čas zamyslet se nad jiným řešením.

Tím řešením se nakonec stala databáze mysql. Rozhodl jsem se tedy ukládat generovaná data do databáze a následně je prezentovat na webových stránkách pomocí kombinace html, php a sql dotazů. Bylo ovšem třeba provést nějaké úpravy.

3.2.1 Skript pro generován dat

Prvním článkem v řetězu prezentace statistik je získání statistických dat. Již v mé předchozí práci jsem předvedl funkční skript pro generování statistických dat. Nyní pouze připomenu jeho základní funkci. Jde o skript pro příkazový interpret bash. Je postaven na programu ping, který vysílá ICMP požadavky na odpověď postupně na všechny IP adresy ze sledovaného seznamu a zachytává ICMP odpovědi, které následně vyhodnocuje. Bohužel z pouze jednoho ICMP požadavku mohu zjistit akorát to, že je server dostupný, nikoliv to že je nedostupný, jelikož by se mohlo stát, že byl paket po cestě náhodou ztracen. Skript je tedy koncipován tak, že lze posílat na jeden server libovolné množství ICMP žádostí. V mém případě jsem běžně používal 10.

Program ping obecně vrací mnoho údajů, které jsou vypisovány na standardní výstup. Při tvorbě tohoto programu autoři zřejmě moc nepočítali s použitím těchto údajů ve skriptech, tudíž je výstup pro skript vcelku nepříjemně formátován. Existuje mnohem vhodnější nástroj *Fping*, který umožňuje formátování výstupu, paralelní zpracování a podobně, ovšem není standardním vybavením většiny linuxových

distribucí a proto se jeho použití raději vyhnu. Po malých úpravách výstupu programu ping pomocí programu grep a awk, se mi však podařilo z výstupu vytáhnout pro mne relevantní údaje, což jsou :

- počet odeslaných ICMP žádostí
- počet přijatých ICMP odpovědí
- Minimální hodnota RTT [1]
- Průměrná hodnota RTT
- Maximální hodnota RTT
- Hodnota Jitter

Z uvedených údajů se může zdát nejasný snad jedině jitter. Je to střední odchylka doby zpoždění. Udává, jak moc se mezi sebou lišily doby zpoždění jednotlivých odpovědí. Je to údaj, který je důležitý zejména pro multimediální přenosy a vzhledem k tomu, že tato práce je součástí projektu, který se zabývá právě multimediálním přenosem, zahrnul jsem jej do statistik také. Přejděme tedy přímo ke skriptu.

V prvé řadě musí skript znát seznam IP adres které má testovat. IP adresy všech uzlů, které existují v planetlabu jsou uloženy v databázi. Jelikož jsem skript psal pro příkazový interpret bash a mysql klient umožňuje interaktivní, ale také neinteraktivní přístup, není problém s komunikací s databází. Vytvořil jsem si tedy funkci, která vykoná jakýkoli SQL dotaz předaný jí jako parametr. Pomocí této funkce, jsem do proměnné uložil seznam všech IP adres v databázi.

```
function sql_exec()
{
    mysql -u $user -p$pass -e "$@" -s $database
}

...
seznam=‘sql_exec "select IP from ADRESY"‘
...
```

Za normálních okolností se při zadání `mysql` spustí myslq klient v interaktivním módu, ale parametr `-e`, klientovi říká, že se má pouze vykonat sql dotaz, který následuje za daným parametrem. Samozřejmě bylo ještě nutno blíže specifikovat použitou databázi a zadat jednak uživatelské jméno a jednak heslo k databázi. Vše je uloženo v příslušných proměnných.

[1] Round-Trip time – doba, která uběhla od vyslání do přijetí paketu.

Nyní již může začít samotný cyklus s odesíláním ICMP požadavků, jejichž výsledky se zpracují a je třeba je někam uložit. Zde se musel původní skript pozměnit tak, aby byl schopen zapisovat data místo do systémů adresářů, do databáze. To nebyl nijak zvlášť složitý úkol, neboť stačilo pouze data rozdělit na jednotlivé prvky následujícím způsobem.

```
i=0;
for item in $vystup; do
    pole[$i]=$item
    i=$(( $i + 1 ))
done
```

Tímto jsem tedy velmi jednoduchým způsobem převedl řetězec, získaný z výstupu programu ping na pole, obsahující jednotlivé položky statistických dat, které se již mohly přímo vložit do databáze pomocí sql dotazu, který je předán výše zmíněné funkci pro vykonání SQL dotazu nad databází.

```
sql_exec "insert into TESTIK
(DATUM, ADRESY_ID, SENT, RECIV, MIN, AVG, MAX, JITT) values
('$datum','$adresa_id','${pole[1]}','${pole[2]}',
'${pole[3]}','${pole[4]}','${pole[5]}','${pole[6]}')"
```

Takto se postupně otestují všechny uzly sítě Planetlab uložené v databázi a výsledky se opět uloží do databáze.

3.2.2 Migrace na databázi

Jak jsem zmínil v minulé kapitole, rozhodl jsem se změnit způsob uložení statistických dat z systémů adresářů na databázi. S tím bylo ovšem spojeno více než jen změna skriptu pro generování dat. V prvé řadě bylo potřeba vytvořit datový model. Z jednoduché úvahy, že vyhledávání textových řetězců, je časově o dost náročnější než vyhledávání celočíselných typů, a že uložení textového řetězce taktéž zabere více místa, jsem se rozhodl vytvořit dvě tabulky. Jednu pro adresy uzlů v planetlabu a jednu pro samotná statistická data. Datový model, stejně jako SQL skript, který jej vytvoří, jsem již představil v kapitole 2.3.2.

Pro uložení IP adresy jsem zvolil varchar s maximální délkou 40 znaků. To z toho důvodu, aby bylo možno v budoucnu ukládat i IP adresy protokolu IPv6. Další volby datových typů snad ani nemá cenu vysvětlovat, vše je zřejmé.

Další věcí, kterou bylo nutno udělat, je import IP adres do databáze. Jelikož je na stránkách planetlabu k dispozici seznam adres pouze ve formě doménových jmen, vytvořil jsem tedy nástroj, který umí převádět doménová jména na IP adresy

a naopak. Jelikož jsem do databáze chtěl ukládat také doménová jména, přidal jsem do skriptu i další funkce. Nyní tedy umí :

- Převádět seznam IP adresy na doménová jména
- Převádět doménová jména na IP adresy
- K IP adrese přidat doménové jméno
- K doménovému jménu přidat IP adresu
- Odfiltrovat neexistující doménová jména resp. IP adresy

Pro uložení do databáze jsem si tedy vytvořil soubor, kde byla na jednom řádku IP adresa a doménové jméno serveru. Bylo třeba vytvořit nástroj, kterým adresy dostat do databáze. Zde ovšem přišlo zamyšlení nad tím, zda je bash zrovna tím nevhodnějším kandidátem na práci z databází. Vzhledem k tomu, že pro každý řádek by se muselo znova a znova navazovat spojení s databází, jsem se rozhodl že pro tento případ použili skriptovací jazyk PHP. Ten je oproti bashi znatelně rychlejší, má k dispozici funkce pro práci s databází a větší funkční arzenál pro práci s texty, přeci jenom byl napsán pro tvorbu dynamického webu.

Pomocí tohoto skriptu jsem tedy nahrál do databáze IP adresy. Ovšem co dál ? Nyní by již mohlo být spuštěno monitorování, ovšem přišel bych o data za bezmála půl roku monitorování, což by byla škoda. Vytvořil jsem tedy další skript (opět pomocí php), kterým jsem byl schopen nahrát veškerá doposud sesbíraná data do databáze. Tento skript procházel adresářovou strukturu vytvořenou předchozím skriptem pro generování dat a každý soubor procházel řádek po řádku (ovšem s výjimkou hardlinků . a .. a souborů těch adres, které již nebyly aktivní, tedy nebyly v databázi) a posílal data do databáze.

V tomto bodě již bylo vše připraveno a celá migrace na databázi byla dokončena.

3.2.3 Prezentace statistik

Pro splnění posledního cíle, který jsem si stanovil, chybí již pouze prezentace výsledků monitoringu. Jak je již zřejmé rozhodl jsem se pro webovou prezentaci. Jelikož není v moci jazyka HTML tvorba grafů ani dotazovaná na databázi, bylo potřeba pro tyto účely použít PHP.

Rozhodl jsem se vytvořit jednoduchou stránku se vstupním formulářem, který bude obsahovat pouze dvě položky. Jedním z nich by měl být prvek `select`, kterému se říká většinou rozbalovací seznam a tlačítko pro odeslání údajů. Select by měl obsahovat všechny IP adresy načtené z databáze a nějaké další položky, které budou

reprezentovat ostatní statistiky. Nakonec jsem se rozhodl pro dvě další položky, jedna bude představovat celkovou statistiku planetlabu, resp. vývoj počtu dostupných a nedostupných serverů a druhá by měla představovat všechny platné IP adresy v planetlabu.

Prvním krokem tedy bylo načtení dat do selectu. V tomto bodě jsem se rozhodl oddělit část pro práci s databází do odděleného skriptu, který budu následně includovat do skriptů ve kterých bude potřeba práce s databází. tento skript jsem nazval `dbquery.php` a obsahuje konstanty, jež reprezentují jednotlivé SQL dotazy. Dále je zde funkce, která vykoná SQL dotaz nad databází a vrátí případný výsledek. Tato funkce vypadá takto následovně.

```
function &db_dotaz($dotaz)
{
    mysql_connect("localhost", "user", "passwd");
    mysql_select_db("database");
    $result=mysql_query($dotaz);
    mysql_close();
    return $result;
}
```

Jednoduše vytvoří spojení s databází, provede dotaz, který je funkci předán jako parametr, uzavře spojení s databázi a vrátí ukazatel na výsledná data.

Tím je tedy vytvořeno rozhraní pro práci s databázi a dále se tedy nemusím starat o rutiny jako je připojování a odpojování od databáze a tvorba SQL dotazů, jelikož ty již mám předem vytvořeny. Tento způsob tvorby SQL dotazů se nehodí úplně vždy. Mnohdy je mnohem lepší si vytvořit jakousi funkci která podle předaných parametrů sama dotaz vygeneruje avšak v tomto případě, kdy dotazů není potřeba tolik je tento způsob dostačující a přehledný.

Nyní již je select naplněn daty, resp. IP adresami z databáze. Přidal jsem také dvě již zmiňované položky. Nyní přichází na řadu otázka co se bude dít při odeslání dat z formuláře. Stránka bude v prvé řadě obnovena. Skript tedy zachytí nastavenou proměnnou (ta byla nastavena odesláním formuláře a je naplněna hodnotou odpovídající vybrané položce) a z bezpečnostních důvodů na ní aplikuje funkci mysql_real_escape_string(), která pomocí zpětného lomítka zneutralizuje případné nebezpečné znaky (s ohledem na použitou znakovou sadu), které by mohly mít za následek úspěšný útok zvaný SQL Injection.

Přichází na řadu vyhodnocení proměnné, aby skript věděl co dále provést. Vzhledem k předchozím rozhodnutím, jsou možné pouze tři stavy :

- Proměnná je IP adresa a je tedy nutno vygenerovat její statistiku
- Proměnná reprezentuje globální statistiku Planetlabu

- Proměnná reprezentuje statistiku všech uzlů

Na toto rozhodnutí jsem použil příkaz switch. Teď již přichází na řadu samotná prezentace kýžených statistik. Pro větší přehlednost jsem vytvořil nový skript jménem `gengraf.php`, který bude obsahovat veškeré funkce, starající se o generování statistik. V tomto skriptu je tedy soustředěna veškerá logika webové prezentace.

Je zřejmé že součástí oněch prezentací statistik budou i grafy. V mé minulé práci *Skripty pro instalaci a konfiguraci operačního systému Fedora*, jsem představil program Gnuplot, pomocí kterého jsem již tehdy vytvořil několik grafů. Tento program hodlám použít i nadále, ovšem neexistuje pro něj žádná funkce v PHP, která by mi to usnadnila. Samozřejmě existují třídy, vytvořené nadšenci, které se tváří jako jednoduché API k programu gnuplot, ovšem žádnou knihovní funkci jsem nenalezl. Rozhodl jsem se tedy nepoužít žádného API a pracovat s programem Gnuplot tak, jak jsem byl doposud zvyklý, tady sám si napsat konfigurační skript.

K tomu, aby gnuplot mohl vykreslit graf, potřebuje dvě věci. Jednak konfigurační skript, který mu říká jak má vypadat výsledný graf, co vše má obsahovat případně kde se má uložit výsledný obrázek a kde najde zdrojová data a jednak samotná zdrojová data. Jelikož zdrojová data není problém vytáhnout z databáze, zbývá pouze vytvořit konfigurační skript.

Obojí, jak konfigurační skript, tak zdrojová data, jsem se rozhodl ukládat do dočasného adresáře operačního systému linux tedy /tmp. Vše se odehrává ve funkcích v souboru `gengraf.php`, který jsem zmínil výše. Zdrojová data jsou tedy pomocí SQL dotazu vytažena z databáze a uložena do souboru v adresáři /tmp a to následujícím způsobem (jen podotknu, že se jedná o ukázku z funkce pro generování statistik jednotlivých uzlů).

```
$vys=db_dotaz(SQL_NODE);
$tmpdat=tempnam("/tmp","graf.dat");
$fd=fopen($tmpdat,"w") or die("nelze otevrit soubor");
while ($zaznam=mysql_fetch_array($vys))
{
    fwrite($fd, $zaznam["DATUM"]." ".$zaznam["SENT"]." "
    .$zaznam["RECIV"]." ".$zaznam["AVG"]." "
    .$zaznam["JITT"]."\n");
}
fclose($fd);
```

Konfigurační data jsou pak přímo ze skriptu upravena na míru pro daný uzel a taktéž jsou uložena do adresáře /tmp. Následně je pak pomocí funkce `exec` spuštěn program gnuplot s parametrem, tedy cestou ke konfiguračnímu souboru. Program

gnuplot již pak vykoná své a vygeneruje obrázky, které se následně zobrazí na webové stránce.

Vzhledem k tomu, že celkový počet adres uzlů je v databázi něco kolem 800 a ke každému se generují tři grafy, navíc data pro celkovou statistiku planetlabu se počítají ze záznamů za celou dobu monitoringu a je tedy celkem časově náročné je získat, ulehčil jsem databázi tím, že každý graf bude vygenerován jenom jednou za určitou dobu, která je specifikována v konstantě `REFRESH` ve skriptu. Vždy při spuštění funkce, tedy požadavku na nějaké grafy, je zkontrolována existence daného grafu v adresáři ./tmp (jehož existence je mimochodem také kontrolována a není tedy problém data promazat) a pokud existuje, je zkontrolováno časové razítko. Pokud je soubor starší než doba uvedená v konstantě `REFRESH` jsou grafy vytvořeny znova, v opačném případě, jsou prezentovány grafy již vytvořené. Celý mechanizmus kontroly tedy vypadá takto.

```
    // doba po kterou zustanou vygenerovane grafy
define("REFRESH", 86400);

    // overeni existence adresare /tmp, pripadne se vytvori
if (!(file_exists("./tmp")) or !(is_dir("./tmp")))
    mkdir("./tmp") or die("nelze vytvort adresar tmp");

    // Osetreni existence podadresare s grafem
if (!(file_exists("./tmp/$item")) or !(is_dir("./tmp/$item")))
    mkdir("./tmp/$item") or die("nelze vytvort adresar tmp/$item");

    // osetreni existence souboru s grafem
if (file_exists("./tmp/$item/av$item.png"))

        // hlida casove razitko souboru a pokud je
        // mladsi nez REFRESH pouzije stavajici soub
    if ((time()-REFRESH) < filectime("./tmp/$item/av$item.png")) {
        print "<img src=\"./tmp/$item/av$item.png\">\n";
        print "<img src=\"./tmp/$item/rtt$item.png\">\n";
        print "<img src=\"./tmp/$item/jit$item.png\">\n";
        return;
    }
```

Tento příklad je z funkce pro generování statistik jednotlivých uzlů. Je to mírně odlišné od toho co se odehrává ve funkci pro generování celkové statistiky Planetlabu, jelikož tam není třeba kontrolovat existenci podadresáře v ./tmp.

Jak jsem již zmínil, funkcí je v souboru `gengraf.php` více. Je to proto, že jsem rozdělil také generování grafů a výpis informační tabulky, ale to je v podstatě pouze kosmetickou záležitostí, která slouží pouze k lepší přehlednosti.

Poslední funkcí, o které jsem se zatím vůbec nezmínil je `getAllNodes()`, což je funkce pro vykreslení tabulky s spolehlivostí všech uzlů v Planetlabu. Tato funkce po načte data, která zahrnují pouze IP adresu uzlu, jeho doménové jméno a vypočtenou spolehlivost. Spolehlivost se vypočítá jako poměr všech obdržených ICMP odpovědí, ke všem odeslaným ICMP žádostem vynásobený stem a vyjadřuje procentuální dosažitelnost serveru. Data jsou pak seřazena sestupně podle této spolehlivosti. Ke každému řádku je také vykreslen pruh, který svou délkou a svým zbarvením graficky reprezentuje hodnotu spolehlivosti. Tedy od nejdelšího zeleného pruhu, až z nejkratšímu červenému pruhu (to je ovšem nepřesné, jelikož nejkratší délka je 0 a pruh tedy není vidět. Ovšem vězte, že kdyby šel vidět, byl by červený).

Tím jsou stránky pro prezentaci výsledků monitoringu hotové a připravené pro použití. Stránka samozřejmě obsahuje krátkou nápovědu, popis a jsou zde dokonce ke stažení i některé skripty, které jsem v této práci vytvořil. Stránka je v době tvorby této práce dostupná z http://adela.utko.feec.vutbr.cz/planetlab_stats/.

4 DISKUZE VÝSLEDKŮ

V této práci jsem si stanovil dva základní cíle. Jedním z nich byla tvorba nástrojů pro konfiguraci operačního systému Fedora v síti Planetlab a druhým byla monitorování zdraví Planetlabu a prezentace těchto výsledků formou webových stránek s grafy

První cíl jsem se snažil dosáhnout tvorbou dvou základních nástrojů resp. skriptů. První skript, nyní ho již můžu pojmenovat `scp_copy.sh` je postaven na programu scp a slouží ke kopírování souborů na servery v Planetlabu, ale také ze serverů. Samotný tento skript, by ještě úplně nesplnil požadavek konfigurovat operační systém Fedora. Zkrátka by to pouze pomocí tohoto cíle bylo velmi obtížné, ne-li nemožné zvládnou. Vytvořil jsem tedy ještě další nástroj, nyní ho již mohu pojmenovat `ssh_execute.sh`. Tento skript je postaven na programu ssh a slouží ke vzdálenému spouštění programů, provádění příkazů, popřípadě i celých skriptů.

Součinností obou skriptů se uživatelům do ruky dostává nástroj, který skýtá téměř neomezené možnosti konfigurace a obecně hromadné správy serverů v Planetlabu. Lze s jejich pomocí spouštět vlastní programy, přesouvat vytvořené soubory, čí konfigurovat prostředí podle potřeb uživatelů.

Oba skripty jsou samozřejmě pouze nástroji, které uživatel dostává do rukou a jejich použití je již pouze na něm. Abych ovšem neznalého uživatel neponechal na holičkách, oba skripty jsou vybaveny obsáhlou nápovědou a jejich zdrojový kód je také dobře okomentován, což značně snižuje možnosti nesprávného použití skriptů, nebo jejich úplné nepochopení. První cíl se mi tedy podařilo beze zbytku naplnit. Oba skripty byly odzkoušeny nejen mnou, ale byli již použity i některými ostatními řešiteli projektu. Prozatím se neprojevily žádné stížnosti na funkčnost či komfortnost skriptu, ale v případě, že nějaké vyvstanou, nebudu váhat s nápravou tak, aby byly skripty i nadále dobře použitelné. Je tedy možné, že se skripty uvedené v této práci mohou časem změnit.

Můj druhý cíl jsem začal zpracovávat již v mé předchozí práci. Výsledky mé minulé práce bylo ovšem nutno předělat podle nově vzniklých potřeb. Hlavní změnou bylo použití databáze jako úložiště statistik. Upravil jsem tedy skript pro generování statistik tak, aby byl schopen generovat data přímo do databáze. Bylo ovšem třeba vytvořit více. Celá migrace stávajících statistik a generování grafů si vyžádala další sadu skriptů. pomocí kterých jsem byl schopen přenést stávající data do databáze.

Vznikl tedy skript pro převod mezi IP adresami a doménovými jmény `add_dns.sh`, pomocí kterého jsem mohl vytvořit soubor vhodný pro import do databáze. Dále pak vznikl skript který takto vytvořená data do databáze nahrál `mysql_adresy.php` a hlavně také skript, který veškeré původní statistiky uložené v souborech převedl do databáze `mysql_import.php`. Tímto byly vytvořeno zázemí pro prezentaci výsledků monitoringu.

Nakonec jsem pomocí html a php vytvořil webovou prezentaci, která pomocí dat z databáze prezentuje výsledky monitoringu planetlabu a to jak formou tabulek tak formou grafů. Grafy jsou generovány pomocí programu gnuplot, který je spouště přímo z php skriptů v závislosti na požadavcích zadaných na webových stránkách. Stránky jsou tedy dynamické, a veškeré informace odrážejí skutečný stav planetlabu a jednotlivých uzlů v něm.

Tímto se mi tedy podařilo splnit i druhý a zároveň poslední cíl této práce a mohu spokojeně prohlásit, že všechny cíle, které jsem si v této práci na počátku určil jsou beze zbytku splněny, a že nástroje, které jsem v práci vytvořil jsou všechny v provozu a plně funkční.

5 ZÁVĚR

Tato bakalářská práce je dílem, které jsem vytvořil vlastními silami, s přispěním informací knih, internetových článků a diskuzních fór. V první části práce jsem uvedl některé teoretické poznatky, které se staly nezbytné pro úspěšné splnění cílů v této práci stanovených. Byly to zejména poznatky týkající se databází, skriptovacího jazyku PHP a v neposlední řadě také samotného operačního systému linux, ve kterém probíhal i samotný vývoj. Bez těchto poznatků by bylo jen stěží možné tuto práci dokončit.

V druhé části jsem se zaměřil na praktickou část a samotnou tvorbu skriptů, které jsou nakonec všechny obsaženy v příloze, snad kromě webové prezentace, která je vystavena na stránkách http://adela.utko.feec.vutbr.cz/planetlab_stats/. Zdrojové kódy této prezentace jsou k dispozici pouze v elektronické příloze, stejně jako i všechny ostatní.

S radostí mohu konstatovat, že všech cílů v této práci bylo dosaženo a funkčnost všech nástrojů byla ověřena v praktické použití v prostředí pro které byly vytvořeny tedy v Planetlabu a skript pro generování statistik v této chvíli běží na serveru adela.utko.vutbr.cz a generuje další statistická data, která jsou pak dostupná z výše uvedených webových stránek.

REFERENCE

[1] BOLDIŠ, P. *Bibliografické citace dokumentů podle ČSN ISO 690 a ČSN ISO 690-2* [online]. 2001, poslední aktualizace 11. 11. 2004 [cit. 17. 2. 2005]. Dostupné z URL: <http://www.boldis.cz/citace/citace.html>.

[2] LACKO, L. *PHP a MySQL - Hotová řešení.* První vydání. Brno: Computer Press, 2005. 300 s. ISBN: 80-251-0397-8

[3] MYSLIK, V. *Filosofie Unixu* [online]. 1996, posledn9 aktualizace 2006[cit. 2.4. 2008]. Dostupné z URL: <http://aldebaran.feld.cvut.cz/ xmyslik/www/nr2.html>.

[4] ROSERBROCK, E. FILSON, E. *Linux, Apache, MySQL a PHP.* První vydání. Praha: Grada, 2005. 344 s. ISBN: 80-247-1260-1

[5] SHNEIDER, R. *MySQL - Oficiální průvodce tvorbou, správou a laděním databáze.* První vydání. Praha: MySQL Press, 2006. 372 s. ISBN: 80-247-1516-3.

[6] ZAJÍC, P. Seriál o MySQL [online]. ISSN 1801-3805 Dostupné z URL: <http://www.linuxsoft.cz/article_list.php?id_kategory=232>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

Bash Bourne Again SHell – Bournův příkazový interpret.

SSH Secure Shell – šifrovaný klient/server protokol v síti TCP/IP.

OpenSSH OpenBSD Secure Shell – soubor programů zprostředkovávajích spojení pomocí protokolu SSH.

ssh Secure Shell – program (klient), který implemntuje protokol SSH.

SCP Secure Copy – protokol pro přenos šifrovaných dat.

scp Secure Copy – program (klient), který implmentuje protokol SCP.

cron Unixový plánovač úloh

crond Deamon pro spouštění naplánovaných úloh.

GPL GNU General Public Licence – všeobecná veřejná licence GNU.

LAMP Kombinace Linux + Apache + MySQL + PHP (Perl, Python) používaná pro webový server.

GNU Rekurzivní akronym z GNU's Not Unix – GNU není Unix.

PHP Rekurzivní akronym z PHP: Hypertext preprocesor – PHP: hypertextový preprocesor; Jde o skriptovací programovací jazyk.

GNU/Linux GNU/Linux – jediný správný název tohoto operačního systému, jelikož Linux samotný je pouze název jádra. Z hnutí GNU pak pochází překladač a většina systémových komponent. Dnes je ovšem většinou nazýván jen linux.

unix-like Operační systém unixového typu – takový operační systém, který se snaží o co největší podobnost s Unixem a jeho filosofií. Např. GNU/Linux, FreeBSD, či dokonce Mac OS X.

PDP-7 Minipočítač od firmy Digital Equipment Corporation.

GECOS General Comprehensive Operating System.

BSD Berkeley Software Distribution – odvozenina Unixu distribuovaná univerzitou Berkley v Kalifornii.

USL Unix System Laboratories – Americká softwarová společnost.

GNU/HURD HURD znamená Hird of Unix-Replacing Daemons a Hird znamená Hurd of Interfaces Representing Depth – je to v podstatě kolekce serverů běžícíhc na mikrokernelu, implementujících jednotlivé části OS. Náhrada Unixového kernelu.

AIX Advanced Interactive eXecutive – propertiální unixový operační systém firmy IBM.

PID Process identifier – identifikační číslo procesu.

RHEL RedHat Enterprise Linux – komerční sitribubce firmy RedHat.

SQL Structured Query Language – strukturovaný dotazovací jazyk pro práci s daty v relačních databázích.

GUI Graphical User Interface – grafické uživatelské rozgraní

PHP PHP: Hypertext Preprocesor – PHP: hypertextový preprocesor; Původní název Personal Home Page

icmp Internet Control Message Protocol – internetový protokol pro přenos chybových a řídících zpráv mezi entitami v síti.

ASF The Apache Software Foundation

NCSA Nationac Center fo Supercomputing Aplications – Národní středisko superpočítačových aplikací

AL Apache Licence – licence kompatibilní s GPL.

RTT Round-Trip time – doba, která uběhla od vyslání do přijetí paketu.

SEZNAM PŘÍLOH

A KONFIGURAČNÍ SKRIPTY

A.1 Skript pro kopírování

```
#!bin/bash
#   Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>
#   Soubor : scp_copy.sh
#   Datum  : 29.11.2007
#   Licence : Vytvoreno jako soucast semestralniho projektu na FEKT VUT v Brne
#
#   Popis : Skript pro kopirovani z/na servery.

#------------------------------INICIALIZACE PROMENNYCH-------------------------
username="cesnet_vutbr1"        # uzivatelske jmeno pro pripojeni na server
log="${0}.log"              # logovaci soubor
unreachable="unreachable.txt"       # soubor s nedostupnymi servery
NEUSPESNE=0

    # promenna obsahujici chybove hlasky
err=(
    "err0 : Zadano prilis mnoho parametru. Parametry navic budou ignorovany.
    Pro napovedu pouzijte prepivac -h"
    "err1 : Zadano malo parametru"
    "err2 : Zadany spatne parametru"
    "err3 : Chybny 1. soubor (soubor neexistuje, nebo ma nulovou velikost)"
    "err4 : Chybny 2. soubor (soubor neexistuje, nebo ma nulovou velikost)"
    "err5 : Server nedostupny"
    "err6 : Chyba pri kopirovani"
    "err7 : Chybny 1. parametr"
    ) # err

#------------------------------POUZIVANE FUNKCE--------------------------------
# funkce na standardni vystup vytiskne napovehu
function print_help()
{
    echo " Skript pro upload souboru (adresaru) na nekolik serveru
    pomoci programu scp.
    Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>

    Popis : Skript stahuje ci kopiruje soubory z nebo na server,
        jejichz seznam se pripojuje jako prvni parametr,
        soubory pomoci programu scp. Pro stahovani ze serveru se
        pouziva prepinac -D. Pro Upload na servery se pouziva prepinac -U.
        V pripade stahovani souboru jsou v cilovem adresari vytvoreny
        adresare pojmenovane jako IP serveru ze kterych se do techto
        adresaru soubory stahuji.
        Zdrojove a cilove adresy jsou v souboru ktery se pripojuje jako
        druhy parametr.

        Format seznamu souboru:
        [zdroj][mezera][cil][mezera][prepinac]

        samozrejme bez hranatych zavorek. Pozn. mezera muze byt
        nahrazena jakymkoli bilym znakem krome konce radku a cilova
        adresa nemusi existovat ale musi byt zadana jako adresar!
        Prepinac neni povinny parametr a pouziva se predevsim -r
        pro rekurzivni kopirovani adresaru.

    Pouziti : "$0" [ -h ] [-D|-U file1 ] [ file2 ]
    Popis parametru :
        [ -h | --help] Vypise tuto napovedu a bezchybne skonci
        [-D]        Prepinac pro stahovani souboru ze serveru
[-U]        Prepinac pro kopirovani souboru na servery
[ file1 ]   Soubor obsahujici adresy serveru ze kterych se
                ma stahovat
        [ file2 ]   Soubor obsahujici adresy souboru ktere se maji
                stahovat a adresy kam se maji stahovane soubory
```

```
                ukladat. Zdroj a cil musi byt na jednom radku
                oddelen alespon jednou mezerou."
    exit 0
} # print_help

    # na standardni vystup vztiskne chybove hlaseni
    # jehoz cislo dostane v 1. parametru
function print_err()
{
    [ $1 -ne 0 ] && echo "${err[$1]}" && print_help
    echo "${err[$1]}"
} # print_err

    # funkce kontroluje dostupnost dane adresy a vraci 0 pokud je dostupna
    # a 1 pokud dostupna neni. Adresa se predava jako prvni parametr
function dostupnost()
{
        # kontrola dostupnosti pingem
    ping -c 1 $1 &> /dev/null
} # dostupnost

    # fukнce obstaravajici chzbovy vystup tykajici se samotneho scp
function failed()
{
    echo "${err[$1]} : udalost ulozena do $log"
    echo "[`date -u +%Y%m%d%H%M%S`] ${err[$1]} : $2 $3 $4" >> $log
    echo $2 >> $unreachable
    NEUSPESNE=`expr $NEUSPESNE + 1`
} # failed

    # funkce spravujici udalosti po uspesne skoncenem kopirovani ze serveru
function success()
{
    echo "[`date -u +%Y%m%d%H%M%S`] Kopirovani souboru $predlozka $1 " \
    "probehlo v poradku" >>$log
    echo "Kopirovani souboru $predlozka $username@$ip probehlo v poradku"
} # Success

    # funkce pro stahovani souboru ze serveru
function download()
{
        # vytvori cilovy adresar, pokud neexistuje
    mkdir -p $3$1/

        # samotne kopirovani souboru
    scp $4 $username@$1:$2 $3$1/ 2>> $log

        # kontrola zda kopirovani probehlo bez problemu
        # pokud ne cyklus se ukonci
    [ $? -ne 0 ] && failed 6 $1  && return 1

     return 0
} # download()

    # funkce pro nahravani souboru na servery
function upload()
{
        # samotne kopirovani souboru
    scp $4  $2 $username@$1:$3 2>> $log

        # kontrola zda kopirovani probehlo bez problemu
        # pokud ne cyklus se ukonci
    [ $? -ne 0 ] && failed 6 $1  && return 1

    return 0
} # upload()
#----------------------KONEC BLOKU PRO DEFINICI FUNKCI-------------------------
```

```
#---------------------------HLAVNI PROGRAM-------------------------------------
    # smazani souboru pro nedostupne servery (bez hlasky pokud neexistuje)
rm -f $unreachable

    # kontrola mnozstvi parametru
[ $# -eq 0 ] &&  print_err 1

    # nastaveni promennych podle prvniho parametru
case "$1" in
    -h )    print_help;;        # prepinac -h vypise napovedu
    -D )    funkce=1        # prepinad -D pro Download ze serveru
            predlozka="z"       # predlozka pro hlaseni vysledku
    ;;
    -U )    funkce=0        # prepinac -U pro Upload na server
        predlozka="na"      # predlozka pro hlaseni vysledku
    ;;
    * )     print_err 7;;       # chybny parametr
esac

    # kontrola mnozstvi parametru
[ $# -lt  2 ] && print_err 1        # malo parametru
[ $# -gt 3 ] && print_err 0     # mnoho parametru, neni ukoncujici chyba


    # kontrola zda soubory pripojene jako parametry existuji
[ -s $2 ] || print_err 3
[ -s $3 ] || print_err 4

    # zaloha promenne IFS
IFS_bak=$IFS

#---------------------------CYKLUS KOPIROVANI NA SERVERY------------------------
for ip in `cat $2`; do
echo "Kopiruji soubory $predlozka $username@$ip :"

        # nastaveni promenne IFS
    IFS=$'\n'
        # cyklus pro kopirovani souboru
    for soubor in `cat $3`; do

            # nastaveni promenn IFS na puvodni hodnotu
        IFS=$IFS_bak

            # rozdeleni radku souboru na jednotlive promenne
        zdroj=`echo $soubor | awk '{print $1}'`
        cil=`echo $soubor | awk '{print $2}'`
        prepinac=`echo $soubor | awk '{print $3}'`

            # kontrola dostupnsti serveru, pokud je nedostupny
            # smycka se ukonci
        dostupnost $ip
        if [ $? -ne 0 ]; then
                failed 5 $ip
                stat=1
                break
        fi

            # volani fce pro Stahovani, nebo Nahravani souboru
            # zalezi na prvnim parametru (-U 0, -D 1)
        if [ $funkce -eq 1 ]; then
                download $ip $zdroj $cil $prepinac
                stat=$?
        else
                upload $ip $zdroj $cil $prepinac
                stat=$?
        fi
            # navratova hodnota funkce Download/Upload

        IFS=$'\n'
    done # for soubor in `cat #2`
```

```
        # v pripade standartniho ukonceni se vzpise hlaska o uspesnem
        # prekopirovani souboru
    if [ $stat -eq 0 ]; then
         success $username@$ip
    fi

    IFS=$IFS_bak
done # for ip in `cat $1`

#------------------------------UKONCENI SKRIPTU--------------------------------

    # Statistika uspesnosti kopirovani
[ $NEUSPESNE -ne 0 ] && echo "Na nekterych serverech probehlo kopirovani "\
"neuspesne viz. unreachable.txt. Pocet neuspesnych kopirovani : $NEUSPESNE" || \
echo "Veskere kopirovani probehlo uspesne"

    # smaze vsechny docasne soubory v pracovnim adresari
rm -f *~ *.bak
echo "Konec skriptu.. "
    # bezchybne ukonceni skriptu
exit 0
```

A.2 Skript pro konfiguraci

```
#!/bin/bash
#       Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>
#       Soubor : ping_skript.sh
#       Datum  : 20.3.2008
#       Licence : Vytvoreno jako soucast semestralniho projektu na FEKT VUT v Brne
#
#       Popis : Skript pro vzdalenou konfiguraci serverů.

#-------------------------------KONSTANTY-----------------------------------

username="cesnet_vutbr1"                # uzivatelske jmeno pro pripojeni na server
log="${0}.log"                          # logovaci soubor
unreachable="unreachable.txt"           # soubor s nedostupnymi servery
skript=0

        # promenna obsahujici chybove hlasky
err=(
        "err0 : Zadano prilis mnoho parametru. Parametry navic budou ignorovany.
        Pro napovedu pouzijte prepivac -h"
        "err1 : Zadano malo parametru"
        "err2 : Zadany spatne parametru"
        "err3 : Chybny soubor s adresami"
        "err4 : Chybny soubor se skriptem"
        "err5 : Server nedostupny"
        "err6 : Pomocny skript skoncil s chybou"

        ) # err


#-----------------------------POUZIVANE FUNKCE--------------------------------
        # funkce na standardni vystup vytiskne napovehu
function print_help()
{
        echo "  Skript pro vzdalenou konfiguraci serverů.

        Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>

        Popis : Spusti zadany prikaz na vzdalenych strojich definovanych v seznamu
                zadavanem jako prvni parametr. Misto prikazu lze pouzit i skripty,
                ktere jsou ulozeny v ./scripts/. V pripade tvorby vlastnich skriptu
                nezapomente na konec vlozit \"rm -f \$0\", aby se na disku
                nehromadily zbytecne soubory. Jakoukoli cinnost lze spustit i na pozadi,
                tak aby bezela i nadale po odpojeni od serveru.

        Pouziti : "$0" [ -h ] [ file1 ] [ -s ] [ -b ] [ command|script ]
        Popis parametru :
                [ -h | --help]  Vypise tuto napovedu a bezchybne skonci
                [ file1 ]       Soubor obsahujici adresy cilovzch serveru
                [ -s ]          Parametr, ktery rika ze se ma pouzit hotovy skript
                                z adresare ./scripts/
                [ -b ]          Parametr spusti danou akci na vzdalenem stroji na
                                pozadi a uzavre standardni vystup, vstup a chybovy
                                vystup, aby akce pokracovala i po odpojeni
                [ command|script ]
                                Muze byt jednotlivy prikaz s parametry, nebo  i vice
                                prikazu spojenych rourami, ci oddelenych strednikem,
                                nebo skript ulozeny v ./scripts/, v tomto pripade
                                zadavat bez pripony a pouzit prepinac -s

                Priklad :

                        sh ssh_script.sh adresy.txt uname -a  -- vypise informace o
                        kazdem cilovem stroji definovanem v souboru adresy.txt

                        sh ssh_script.sh adresy.txt -s -b calculate -- spusti skript
                        calculate z adresare ./scripts/ a necha jej bezet i po
                        odpojeni
```

```
                Poznamka : Opatrne pri spousteni skriptu na pozadi, za vsech
                        okolnosti se bude hlasit spravne provedeny skript
                        a to i v pripade fatalni chyby ve spoustene akci."

        exit 0
} # print_help

        # vytiskne na stdout chybovou hlasku a ukonci skript
function print_err()
{
        [ $1 -ne 0 ] && echo "${err[$1]}" && print_help
        echo "${err[$1]}"
} # print_err

        # kontrola dostupnosti serveru
function dostupnost()
{
                # kontrola dostupnosti pingem
        ping -c 1 $1 &> /dev/null
} # dostupnost

        # Vytiskne na stdout informace o chybe a ulozi do logu
function failed()
{
        echo "${err[$1]} : udalost ulozena do $log"
        echo "[`date -u +%Y%m%d%H%M%S`] FAIL ${err[$1]} : $2" >> $log
        echo $2 >> $unreachable
} # failed

        # vytiskne na stdout info o uspesne akci a ulozi do logu
function success()
{
        echo "[`date -u +%Y%m%d%H%M%S`] OK $1" >>$log
        echo "Skript probehl v poradku"
} # Success


        # V pripade napovedy
if [ "$1" == "-h" ] || [ "$1" == "--help" ]; then print_help; fi

        # kontrola mnozstvi parametru
[ $# -lt  2 ] && print_err 1            # malo parametru


        # kontrola zda soubory s adresami existuje
[ -s $1 ] || print_err 3

rm -f $unreachable
adresy=`cat $1`

        # kontrola pritomnosti parametru -s a -b
shift
while [ 1 -eq 1 ]; do
case "$1" in
        -b )    bckg=" 2<&- 1<&- 0<&- &"        # parametr -b proces bude
                        shift                   # spusten na pozadi
                        continue
                        ;;
        -s )    shift                           # parametr -s bude pouzit
                        skript=1                # externi skript
                        continue
                        ;;
        * )     break
                        ;;
esac
done

        # smycka ve ktere se provadi akce na jednotlivych adresach
for ip in $adresy; do
        echo "Server: $ip"
```

```
                # kontrola dostupnosti
        dostupnost $ip
        [ $? -ne 0 ] && failed 5 $ip && continue

                # Jako parametr byl predan skript
        if [ $skript -ne 0 ]; then

                launch="scripts/${1}.sh"

                        # kontrola yda externi skript existuje
                [ -s $launch ] || print_err 4

                        # Skript je nutno nakopirovat na vydaleny stroj
                scp -r $launch $username@$ip:.$$skript.sh &> /dev/null

                        # Sousteni skriptu na vydalenem stroji
                ssh $username@$ip sh .$$skript.sh $bckg 2>> $log
                [ $? -eq 0 ] && success $ip || failed 6 $ip;

                # Jako parametr byl predan prikaz
        else
                        # Provedeni prikazu na vzdalenem stroji
                ssh $username@$ip $@ $bckg 2>> $log
                [ $? -eq 0 ] && success $ip || failed 6 $ip;
        fi

done
exit 0
```

B SKRIPT PRO GENEROVÁNÍ STATISTIK

```
#!bin/bash
#       Autor : Czerner Lukas <xczern00@stud.feec.vutbr.cz>
#       Soubor : ping_skript.sh
#       Datum  : 29.11.2007
#       Licence : Vytvoreno jako soucast semestralniho projektu na FEKT VUT v Brne
#
#       Popis : Skript odesila icmp zadosti podle danych parametru adresy
#               nactene z databaze, a vysledek profiltruje a ulozi opet do databaze.
#               Prvni parametr udava pocet pingu na jednu adresu.
#               Skript je tvoren pro pridani do cronu, proto neobsahuje
#               zadnou nekonecnou smycku.

#---------------------VYTVORENI POTREBNYCH SOUBORU A ADRESARU-------------------
        # parametry pro pripojeni k databazi
user="user"
pass="pass"
database="db"

        # Funcke vzkona SQL dotaz
function sql_exec()
{
        mysql -u $user -p$pass -e "$@" -s $database
}


#---------------------NASTAVENI POTREBNYCH PROMENNYCH---------------------------
        # parametr udava pocet pingu na jednu adresu v jednom cyklu
pocet=$1

        # seznam IP adres ze souboru (ktery se pripojuje jako 2 parametr)
seznam=‘sql_exec "select IP from ADRESY"‘

        # ulozi aktualni cas a datum do promenne
datum=‘date "+%Y-%m-%d %H:%M:%S"‘

#---------------------SAMOTNY CYKLUS VE KTEREM PROBIHAJI PINGY------------------
for ip in $seznam; do

                # formatovani vystupu pingu do pozadovaneho tvaru
        vystup=‘ping -n -c $pocet -i 0.5 -W 2 $ip |\
                grep -A 2 statistics‘
        vystup=‘echo $vystup | awk ’{print $2 " "  $6 " "  $9 " " $19}’\
                | sed ’y|/| |’‘

        i=0;

                # Cyklus pro rozdeleni vystupu pingu do pole
        for item in $vystup; do
                pole[$i]=$item
                i=$(( $i + 1 ))
        done

                # dotaz na klic v tabulce adresy
        adresa_id=‘sql_exec "select ID from ADRESY where IP=’${pole[0]}’" | tail -n 1‘

                # Vlozeni dat do databaze
        sql_exec "insert into STATISTIKY (DATUM, ADRESY_ID, SENT, RECIV, MIN,
        AVG, MAX, JITT) values (’$datum’,’$adresa_id’,’${pole[1]}’,’${pole[2]}’,
        ’${pole[3]}’,’${pole[4]}’,’${pole[5]}’,’${pole[6]}’)"

        unset pole

done
date -u +%Y%m%d%H%M
        # Odstraneni souboru s cislem procesu
rm -f pid
exit 0
```

C POMOCNÉ SKRIPTY

C.1 Skript pro import adres do databáze

```
<?php
        // Pripojeni k databazi
mysql_connect("localhost", "user", "pass") or die ("Nelze otevrit databazi");
mysql_select_db("db");

$fp = fopen("/cesta/adresy_dns.txt","r") or die ("fOpen selhal");

while (!(fEOF($fp)))
{

        $radek= fGets($fp, 255);

                // rozdeleni polozek z radku na pole
        $prvky=explode(" ","$radek");

                // tvorba sql prikazu
        @$sql="insert into ADRESY (IP, DNS) values ('$prvky[0]','$prvky[1]')";
        mysql_query("$sql");
}
fClose($fp);
mysql_close();
?>
```

C.2 Skript pro import statistik do databáze

```
<?php
        // Pripojeni k databazi
mysql_connect("localhost", "user", "pass") or die ("Nelze otevrit databazi");
mysql_select_db("db");

        // otevreni adresare
if ($handle=opendir('/cesta/ke/statistikam')) {
                // cteni obsahu adresare
        while ($file=readdir($handle))
        {
                        // v seznamu se nesmi obevit hardlinky . a ..
                if ($file != "." && $file != "..")
                {
                                // otevreni souboru pro cteni
                        $fp = fOpen("/cesta/ke/statistikam/$file","r") or die ("fOpen selhala");
                        echo $file."\n";
                        $vys=mysql_fetch_array(mysql_query("select ID from ADRESY where IP='$file'"));
                        if ($vys[0] == "") {
                                        echo "adresa $file nenalezena";
                                        continue;
                        }
                                // smycka prochazeni souboru po radcich
                        while (!(fEOF($fp)))
                        {

                                $radek= fGets($fp, 70);

                                        // rozdeleni polozek z radku na pole
                                $prvky=explode(" ","$radek");


                                        // tvorba sql prikazu
                                @$sql="insert into STATISTIKY (DATUM, ADRESY_ID, SENT, RECIV,
                                MIN, AVG, MAX, JITT) values ('$prvky[0]','$vys[0]','$prvky[2]','
                                $prvky[3]','$prvky[4]','$prvky[5]','$prvky[6]','$prvky[7]')";

                                mysql_query("$sql");
                        }

                                // uzavreni souboru
                        fClose($fp);

                }
        }
                // uzavreni adresare
        closedir($handle);
}

mysql_close();

?>
```

C.3 Skript pro převod IP adres a doménových jmen

```
#!bin/bash
#       Author : Czerner Lukas
#       Filename : add_dns.sh
#       Date  : 3.4.2008
#
#       Licence : Vytvoreno jako soucast projektu na VUT v Brne
#
#       Description : K souboru a IP prida druhy sloupec DNS nazev
#
################################################################################


function add_dns ()
{
        for ip in $adresy ; do
                dns=‘dig +noall +answer -x  $ip | head -n 1 |grep PTR|awk {’print $5’}‘
                if [ "$dns" != "" ]; then
                        printf "$ip $dns\n" | sed ’s/.$//’
                fi
        done
}

function todns ()
{
        for ip in $adresy ; do
                dns=‘dig +noall +answer -x  $ip | head -n 1 |grep PTR|awk {’print $5’}‘
                if [ "$dns" != "" ]; then
                        printf "$dns\n" | sed ’s/.$//’
                fi
        done
}

function add_ip()
{
        for dns in $adresy ; do
                ip=‘host -t A $dns | head -n 1 | awk ’{print $4}’‘
                if [[ "$ip" =~ ^[0-9].*[0-9]$ ]]; then
                        printf "$ip $dns\n"
                fi
        done
}

function toip()
{
        for dns in $adresy ; do
                ip=‘host -t A $dns | head -n 1 | awk ’{print $4}’‘
                if [[ "$ip" =~ ^[0-9].*[0-9]$ ]]; then
                        printf "$ip\n"
                fi
        done
}


adresy=‘cat $1‘
case "$2" in
        +dns )
                add_dns
                ;;
        +ip )
                add_ip
                ;;
        -toip )
                toip
                ;;
        -todns )
                todns
```

```
                ;;
        * )
                echo "Zadejte druhy parametr +dns, +ip, -toip, nebo -todns"
                ;;
esac
```