VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

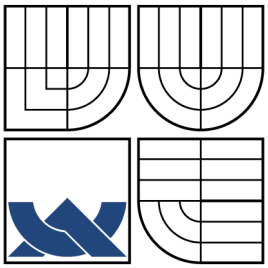
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

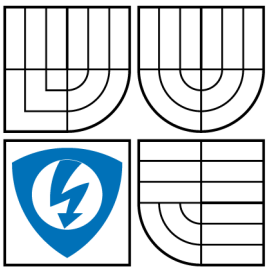
APLIKACE PRO SPRÁVU SERVERŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PETR SMAHEL
AUTHOR



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

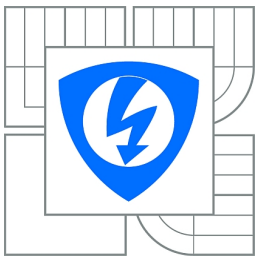
APLIKACE PRO SPRÁVU SERVERŮ
APPLICATION FOR SERVER MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR SMAHEL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ONDŘEJ KRAJSA
SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Petr Smahel	***ID:***	78752
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Aplikace pro správu serverů

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte si možnosti programování aplikací pod operačním systémem Linux. Navrhněte a realizujte aplikaci typu klient-server, která bude monitorovat vzdálený systém (činnost, běžící procesy, přihlášené uživatele, vytížení, atd.) a data předávat ke zpracování serveru. Zpracovaná data budou dostupná ve webovém rozhraní. Zabezpečte systém proti vniknutí neoprávněné osoby.

DOPORUČENÁ LITERATURA:

[1] Dařena, František, Myslíme v jazyku Perl. Praha : Grada, 2005. 700 s. ISBN 80-247-1147-8
[2] Eckel, Bruce, Myslíme v jazyku C++. Praha : Grada, 2000. 554 s. ISBN 80-247-9009-2
[3] PHP 6, MySQL, Apache :vytváříme webové aplikace. Brno : Computer Press, 2009. 816 s. : il. ISBN 978-80-251-2767-4
[4] Veselský, J. LINUX :dokumentační projekt. Brno : Computer Press, 2003. 1001 s. ISBN 80-7226-761-2

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Ondřej Krajsa

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabývá problematikou správy počítačových serverů. První, teoretická část popisuje architekturu klient-server, operační systém GNU/Linux a nástroje pro správu serverů. Druhá část práce se věnuje návrhu vlastní aplikace pro správu serverů. V další části je popsán návrh a implementace jednotlivých modulů, ze kterých se aplikace skládá. Ve čtvrté části této práce je popsána instalace a konfigurace celého systému na platformu GNU/Linux. Předposlední část je koncipována jako uživatelský manuál a seznamuje čtenáře s ovládáním aplikace. V poslední části jsou prezentovány a komentovány výsledky získané při testování aplikace.

KLÍČOVÁ SLOVA

Server, Správa, Linux, Klient, JPGraph, Apache, Stunnel, Aplikace

ABSTRACT

This thesis deals with the management of computer servers. The theoretical part describes the client-server architecture, operating system GNU/Linux and server management tools. The second chapter is devoted to design an own application to manage servers. The next chapter describes the design and implementation of individual modules from which application is build. In the fourth chapter there is described how to install and configure the application on the GNU/Linuw operating system. The penultimate chapter is written as a user manual and acquaints reader with application control. In the last section there are results obtained in testing the application presented and commented.

KEYWORDS

Server, Management, Linux, Klient, JPGraph, Apache, Stunnel, Application

SMAHEL, Petr *Aplikace pro správu serverů*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 80 s. Vedoucí práce byl Ing. Ondřej Krajsa

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Aplikace pro správu serverů“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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.

Brno ....

(podpis autora)

Poděkování

Chtěl bych poděkovat všem lidem, kteří přispěli ke vzniku této práce, zejména pak vedoucímu diplomové práce Ing. Ondřeji Krajsovi za velmi užitečnou metodickou pomoc a cenné rady při zpracování úkolu.

OBSAH

Úvod **8**

1 Teoretické řešení **9**
1.1 Architektura klient-server 9
1.1.1 Komunikace klient-server 10
1.2 Operační systém GNU/Linux 11
1.2.1 Původ GNU/Linuxu 12
1.2.2 Charakteristika 12
1.2.3 Distribuce 12
1.3 Správa linuxových serverů 13
1.4 Monitorování 14
1.4.1 Souborové systémy `/proc` a `/var` 14
1.4.2 Nástroje pro monitorování hardwaru 14
1.4.3 Nástroje pro monitorování zátěže systému 15
1.4.4 Nástroje pro pokročilejší monitorování 17

2 Návrh vlastní aplikace **20**
2.1 Specifikace požadavků 20
2.2 Obecný návrh 20
2.2.1 Stručný popis funkce 21
2.3 Výběr technologií pro implementaci aplikace 21
2.3.1 Klient a server 22
2.3.2 Datové úložiště 22
2.3.3 Webové rozhraní 24
2.3.4 Tvorba grafů 25

3 Implementace **26**
3.1 Klient 26
3.1.1 Soubory obsahující zdrojové kódy klienta 26
3.1.2 Stručný popis činnosti klienta 27
3.1.3 Načtení pluginů 27
3.1.4 Spuštění pluginů 28
3.1.5 Komunikace se serverem 28
3.2 Pluginy 29
3.2.1 Soubory obsahující zdrojové kódy pluginu 30
3.2.2 Formát dat 30
3.2.3 Získání dat 31

3.2.4 Vytvoření vlastního pluginu . . . 31
3.3 Server . . . 33
3.3.1 Soubory obsahující zdrojové kódy serveru . . . 33
3.3.2 Stručný popis činnosti serveru . . . 34
3.3.3 Komunikace s klientem . . . 34
3.3.4 Zpracovaní přijatých dat . . . 36
3.4 Logovaní . . . 36
3.5 Zabezpečení . . . 37
3.6 Databáze . . . 38
3.6.1 Tabulky . . . 38
3.6.2 Přepočítávání . . . 39
3.7 Webové rozhraní . . . 42
3.7.1 Adresářová struktura . . . 42
3.7.2 Prezentace textových dat . . . 43
3.7.3 Prezentace numerických dat . . . 43
3.7.4 Generování grafů . . . 43
3.7.5 Konfigurace grafů . . . 45
3.7.6 Uživatelské účty . . . 45
3.7.7 Zabezpečení . . . 45

4 Instalace a konfigurace **47**
4.1 Instalace operačního systému . . . 47
4.2 Instalace potřebných knihoven a programů . . . 47
4.2.1 Software nutný pro kompilaci . . . 47
4.2.2 MySQL . . . 48
4.2.3 Webový server . . . 48
4.2.4 Knihovna Jpgrah . . . 48
4.2.5 Stunnel . . . 49
4.3 Instalace a konfigurace klienta a serveru . . . 49
4.3.1 Instalační adresáře . . . 50
4.3.2 Cíle pro make . . . 50
4.3.3 Automatický start . . . 50
4.3.4 Konfigurační soubory . . . 51
4.4 Instalace a konfigurace databáze . . . 53
4.4.1 Instalace funkce `median` . . . 54
4.4.2 Plánovač . . . 55
4.4.3 Zálohování . . . 55
4.5 Instalace a konfigurace webového rozhraní . . . 56
4.6 Zabezpečení . . . 56

4.6.1 Konfigurace programu Stunnel . . . 56
4.6.2 Konfigurace HTTPS . . . 58

5 Používání aplikace 60
5.1 Klient a server . . . 60
5.2 Webové rozhraní . . . 60
5.2.1 Přihlášení . . . 60
5.2.2 Rozmístění ovládacích prvků . . . 60
5.2.3 Správa uživatelů . . . 61
5.2.4 Zobrazení grafů . . . 62
5.2.5 Zobrazení textových výpisů . . . 64

6 Testování aplikace 65
6.1 Parametry závěrečného testu . . . 65
6.2 Výsledky testu . . . 65
6.2.1 Grafy . . . 66

7 Závěr 68

Literatura 69

Seznam symbolů, veličin a zkratek 72

Seznam příloh 73

A Obsah přiloženého DVD 74

B Souborový systém `/proc` 75

C Souborový systém `/var` 76

D Třída `client` 77

E Třída `server` 79

ÚVOD

Jedním z hlavním stavebních kamenů počítačových sítí včetně Internetu je architektura klient-server. Servery poskytují klientům data a informace prostřednictvím nejrůznějších služeb. Klient tedy sám o sobě žádnými „znalostmi“ nedisponuje a chce-li je získat, musí si o ně říct serveru, tímto se klient na serveru stává silně závislým. Dojde-li k výpadku nebo poruše serveru, klient se k požadovaným datům nedostane a musí čekat, dokud nebude funkčnost serveru obnovena. Jelikož čas je v dnešní době velmi drahý, je v zájmů každého provozovatele serveru pečovat o svůj systém tak, aby maximálním možným způsobem nefunkčnost serveru nebo jeho služeb eliminoval.

Cílem této práce je popsat problematiku správy linuxových serverů a přiblížit čtenáři metody a nástroje, které jsou k tomuto účelu navrženy. Dále pak navrhnout a realizovat vlastní aplikaci typu klient-server, která bude monitorovat činnost vzdálených systémů, získaná data statisticky zpracovávat a předkládat v přehledné formě k analýze prostřednictvím webových stránek.

Teoretická část této práce se v úvodu věnuje modelu klient-server jako stěžejní architektuře pro síťovou technologii. Jsou zde popsány hlavní části a principy komunikace mezi nimi. Další kapitola je věnována Operačnímu systému GNU/Linux, jenž je hojně využíván pro realizaci serverových řešení. Činnosti spojené se správou serverů jsou rozebrány v předposlední kapitole. Poslední kapitola se věnuje metodám a nástrojům, jež je možné využít pro monitorování serverů. Jsou zde popsány čistě konzolové nástroje, ale také rozsáhlé systémy s grafickým uživatelským rozhraním.

Návrhu vlastní aplikace se věnuje druhá část této práce. Jednotlivé kapitoly popisují požadavky na aplikaci, samotný návrh vlastního systému a volbu technologií pro implementaci jeho částí.

Třetí část je věnována návrhu a implementaci jednotlivých částí aplikace. Je zde podrobně popsán způsob získávání dat ze vzdálených stanic, jejich následné zpracovávání a uchovávání a v neposlední řadě též způsoby, jakými jsou prezentována uživateli.

Instalace vytvořeného celku na platformu Debian GNU/Linux je popsána krok za krokem ve čtvrté části této práce. Dále je zde čtenář seznámen konfiguračními možnostmi aplikace a také s technologiemi zabezpečujícími aplikaci proti vniknutí neoprávněné osoby.

Předposlední, pátá část, je koncipována jako uživatelský manuál. Seznamuje čtenáře s ovládáním aplikace v prostředí příkazové řádky a především pak s webovým rozhraním, které slouží k interpretaci získaných dat.

V poslední části jsou prezentovány a komentovány výsledky získané při závěrečném testování aplikace.

1 TEORETICKÉ ŘEŠENÍ

V této části práce jsou nejprve popsány základní principy síťové architektury klient-server, dále pak operační systém GNU/Linux jakožto platforma, pro kterou je aplikace primárně určena. Další kapitola je věnována teoretickým poznatkům ohledně správy linuxových serverů. Poslední kapitola se zabývá monitorováním serverů a popisuje nástroje, které je možné pro tuto činnost využít.

1.1 Architektura klient-server

Výraz server je odvozen z anglického slova *serve*, což mimo jiné znamená sloužit, obsloužit. Na termín server je v počítačové terminologii možné nahlížet ze dvou úhlů pohledu. Zaprvé obecně jako na fyzický počítač, který poskytuje nějaký souhrn služeb (např. webový server, poštovní server, databázový server), a zadruhé jako na program, který tuto službu realizuje. Serverový program je v unixových systémech označován jako démon (z anglického *daemon*), v Microsoft Windows pak jako služba (z anglického *service*).

Servery poskytují služby klientům. Jako klient se původně označovalo zařízení, které nebylo schopno provozovat své vlastní programy samostatně, ale mohlo komunikovat se vzdáleným serverem (na kterém byla aplikace spuštěna) přes počítačovou síť a aplikace tak pouze zobrazovat.

Na klienta je možné, podobně jako na server, nahlížet obecně jako na počítač využívající komplexních služeb serveru nebo jako na program, který komunikuje se svým serverovým protějškem pomocí protokolu, jemuž obě strany rozumí (např. protokol HTTP pro web, protokol SMTP pro elektronickou poštu).

Komunikace může probíhat buďto lokálně, kdy se server i klient nacházejí na jednom počítači, nebo síťově, kdy jsou servery a klienti rozmístěni v nějaké počítačové síti (např. v Internetu).

Tato komunikační architektura je označována jako klient-server a je jednou z hlavních myšlenek síťové technologie. Na rozdíl od jiných síťových modelů (jako např. peer-to-peer) si v architektuře klient-server stanice nejsou rovny. Mezi hlavní výhody tohoto síťového modelu patří:

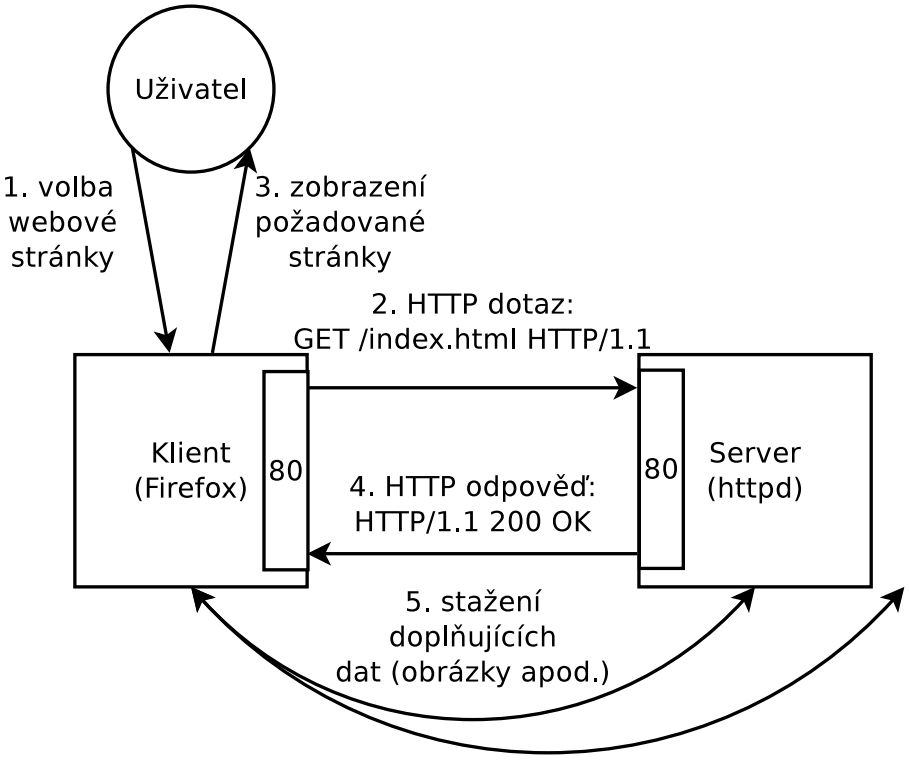
- klient se nemusí starat o mnohdy výpočetně náročné procedury, pouze zpracovává hotová data.
- klient nemusí znát vnitřní strukturu systému, z něhož požaduje data, což má také pozitivní dopad na zabezpečení dat.
- při změně algoritmu by se tato změna musela distribuovat do všech klientů, což není zdaleka vždy reálné, v případě klient-server by mohlo stačit modifikovat pouze serverovou část.

Tato architektura má samozřejmě i své nevýhody:

- při připojení mnoha klientů současně může dojít k přetížení a kolapsu sítě nebo k přetížení serveru.
- při výpadku serveru nebudou požadavky klientů obslouženy a klienti tudíž nedostanou požadovaná data.

1.1.1 Komunikace klient-server

Jak už bylo uvedeno výše, komunikace mezi klientem a serverem probíhá pomocí protokolu, který znají obě strany. Na obrázku 1.1 je vidět průběh komunikace mezi klientem a serverem pomocí protokolu HTTP.



Obr. 1.1: Ukázka komunikace klient-server pomocí protokolu HTTP

Podrobnější popis komunikace klient-server

Webový server httpd naslouchá na portu 80. Jako klient v tomto případě vystupuje online sniffer a jeho požadavkem na server je, aby mu poslal kód požadované stránky (`www.vutbr.cz/index.html`). Nejprve dojde k překladu doménového jména na IP adresu, následuje navázaní TCP spojení a poté je odeslán HTTP požadavek, který je ukončen prázdným řádkem. Hlavička HHTP požadavku je zobrazena ve výpisu 1.1.

```
Connect to 147.229.2.90 on port 80 ... ok

GET /index.html HTTP/1.1
Host: www.vutbr.cz
Connection: close
User-Agent: Web-sniffer/1.0.36 (+http://web-sniffer.net/)
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no
Accept-Language: de,en;q=0.7,en-us;q=0.3
Referer: http://web-sniffer.net/
```

Výpis 1.1: Hlavička HTTP požadavku.

Server načte požadovaný dokument a odešle odpověď. Hlavička obsahuje informaci o tom, že se dotaz podařil, datum a čas vyřízení dotazu, popis serveru, který odpovídá, informace o typu vráceného dokumentu atd. Za hlavičkou je vložen prázdný řádek a pak už následuje samotný zdrojový kód požadované stránky. Hlavička HHTP odpovědi je zobrazena ve výpisu 1.2. Více informací o protokolu HTTP je možné najít v [6].

```
Status: HTTP/1.1 200 OK
Date:   Thu, 18 Nov 2010 12:26:00 GMT
Server: Apache/2.2.3 (Red Hat)
X-Powered-By:   PHP/5.1.6
Expires:        Mon, 01 Jun 1980 00:00:00 GMT
Last-Modified:  Thu, 18 Nov 2010 12:26:01GMT
Cache-Control:  no-cache, must-revalidate
Pragma: no-cache
Connection:     close
Transfer-Encoding:      chunked
Content-Type:   text/html; charset=utf-8

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
               "http://www.w3.org/TR/html4/loose.dtd">
<html>
.
kód stránky
.
</html>
```

Výpis 1.2: Hlavička HTTP odpovědi.

1.2 Operační systém GNU/Linux

GNU/Linux patří do rodiny unixových operačních systémů. GNU/Linux je využíván hlavně v serverových instalacích bez grafického uživatelského rozhraní pro svou stabilitu, možnost snadné vzdálené zprávy a v neposlední řadě pro nízké pořizovací náklady. GNU/Linux si ale pomalu hledá své místo i na počítačích běžných uživatelů, kde jeho podíl v poslední době prudce roste.

1.2.1 Původ GNU/Linuxu

Projekt GNU[1] založil v roce 1983 Richard Stallman[2]. Jeho cílem bylo vytvořit nový operační systém unixového typu, který by byl složen jen ze svobodného softwaru. Za tímto účelem sepsal Stallman novou licenci GNU GPL (GNU General Public License), pod kterou jsou šířeny všechny části systému GNU.

V roce 1991 začal finský student Linus Torvalds[3] pracovat na vývoji vlastního unixového jádra, odvozeného od unixového operačního systému Minix. Jeho jádro, jež nakonec dostalo jméno Linux, si okamžitě našlo řadu příznivců, kteří se začali na jeho vývoji aktivně podílet. Linus se později rozhodl zdrojové kódy uvolnit pod svobodnou licencí GNU GPL.

Spojením GNU, Linuxu a dalších projektů vznikají tzv. distribuce, jež jsou kompilací jednotlivých částí a tvoří tak komplexní operační systém.

1.2.2 Charakteristika

GNU/Linux je především svobodný, sofistikovaný a univerzální systém. Může být nasazen do embedded zařízení, na osobní počítače či servery, ale i na sálové počítače. Byl portován na širokou paletu architektur – od běžné x86 (osobní počítače 386 a vyšší), přes PowerPC , Sparc , Motorola 68k až třeba po MIPS. Podporuje multitasking (běh několika programů současně) a je víceuživatelský (několik uživatelů může současně pracovat na jednom fyzickém systému). Stejně dobře pracuje jak v textovém, tak i v grafickém režimu. Používá moderní žurnálovací souborové systémy, jako například ext3, ReiserFS, XFS.

Pro správu software využívá většina distribucí tzv. balíčkovací systémy. Ty zajišťují nejen instalaci a odinstalaci balíčků, ale řeší i závislosti mezi jednotlivými balíčky. Balíčky se obvykle nachází v tzv. repozitářích (zdrojích softwaru), odkud jej balíčkovací systémy stahují.

1.2.3 Distribuce

Jak již bylo řečeno, GNU/Linux je šířen v distribucích. Distribuce je samotný operační systém GNU/Linux a velké množství aplikací. Existuje řada firem, organizací, skupin i jedinců, kteří sestavují a nabízejí své distribuce. Distribucí jsou v dnešní době stovky. Velmi oblíbenou a na servery hojně nasazovanou distribucí je Debian.

[1] Rekurzivní zkratka vytvořena z prvních písmen anglických slov GNU's Not Unix.

[2] Richard Matthew Stallman (* 16. března 1953, Manhattan, New York), známý též pod iniciálami RMS, je zakladatel hnutí svobodného softwaru, projektu GNU a v říjnu 1985 také Free Software Foundation.

[3] Linus Benedict Torvalds (* 28. prosince 1969) je finský programátor, známý především zahájením vývoje jádra operačního systému Linux a pozdější koordinací projektu.

Debian – je svobodná nekomerční distribuce výhradně ze svobodného softwaru, která je navíc plně vyvíjena komunitou. Debian je znám svým specifickým vývojovým cyklem, kdy jsou k dispozici tři větve: stable, testing a unstable. Všechny softwarové balíčky, které se do distribuce dostanou, postupně prochází testy až do stabilní verze. Díky tomu jsou zde všechny DEB balíčky perfektně odzkoušené, na druhou stranu nejsou nejaktuálnější. Velkou výhodou Debianu je také jeho balíčkovací systém apt, který řídí veškeré instalace a řeší veškeré její problémy a potřebné kroky provádí za uživatele.

Mezi další oblíbené distribuce patří například Ubuntu, OpenSUSE, Mandriva, které se snaží být maximálně uživatelsky přívětivé a jsou určeny především pro nasazení na desktopech, protipólem k nim jsou pak distribuce Slackware, Gentoo, Arch, které jsou určeny pokročilejším uživatelům.

1.3 Správa linuxových serverů

Obecně je úkolem správy zajistit nepřetržitý chod serveru, a tím i dostupnost služeb, které jsou na serveru provozovány. Velmi účinným nástrojem pro správu serverů je linuxová konzole, ve spojení SSH je možné se k serveru vzdáleně přihlásit a spravovat jej prakticky odkudkoliv. Existují i různá grafická řešení, jejichž použití může být v některých případech snadnější a pohodlnější. Správu serverů je možné rozdělit na tři hlavní činnosti.

Konfigurace – konfigurace představuje modifikaci aplikace za účelem získání kýžených vlastností. Potřeby se mění, uživatelé přicházejí a odcházejí, nastupují nové technologie, je nutné sladit více různých aplikací – důvodů je celá řada.

Monitorování – monitorovat server znamená nepřetržitě sledovat stav služeb a systému. Monitoring přináší aktuální a detailní informace pro co největší zkrácení doby problém – řešení. Monitorováním serveru se zjišťuje, zda hardware daným úlohám dostačuje, jestli na disku zbývá dostatek místa, popřípadě zda aplikace funguje tak, jak by měla.

Řešení problémů – služba, která neodpovídá, nedostatek místa na disku, bug[4] vyžadující aktualizaci, vyžadují nápravná opatření ze strany administrátora. Nerozhoduje, jak je systém výkonný či stabilní. Problémy se za jistý čas stejně

[4]Programátorská chyba se často i v češtině označuje anglickým výrazem bug a proces jejího odstraňování ladění (debugování). Bug znamená doslova moucha, štěnice nebo obecně brouk. V angličtině se ve významu chyba (například konstruktérská) používá už velmi dlouho – použil ho například Thomas Edison roku 1878, když mluvil o svých vynálezech. S počítači pak pronikl do mnoha dalších jazyků.

objeví a je na linuxových administrátorech, aby si s nimi poradili a vyřešili je co nejrychleji a nejsnadněji, jelikož samy se nevyřeší [2].

1.4 Monitorování

Monitorování serveru představuje umění zajištění neustálého dohledu nad stavem zařízení a služeb. Jeho cílem je zabezpečení aktualizovaného stavu služeb a dalších prostředků pro jednoduchou diagnózu a řešení problémů. Monitoring je velmi důležitým administrativním úkolem, někdy sice zdlouhavým, ale bezpochyby jediným, který umožňuje provádět přesnou diagnózu, upozorňuje na hrozící nebezpečí a zlepšuje dostupnost služeb [2].

Možností jak monitorovat linuxový server je poměrně hodně. Od manuálního procházení systémových souborů (zejména v `/proc` a `/var`), přes použití nástrojů pro zobrazení nejrůznějších informací o systému (např. `top, free, lm-sensors` apod.), až po využití aplikací, které automaticky zaznamenávají a zpracovávají hlídaná data a umožňují jejich přehledné zobrazení (např. formou grafů).

1.4.1 Souborové systémy `/proc` a `/var`

Souborovým systém `/proc` je vlastně virtuálním souborovým systémem. Ve skutečnosti na disku neexistuje. Místo toho jej v paměti vytváří jádro systému. Původně byl používán pouze pro správu procesů (proto název `proc`). Postupem času bylo do struktury `/proc` přidáno mnoho různých informací. Zásadním rozdílem od jiných unixových systémů je, že každý soubor je textový a uživatel dokáže přečíst jeho obsah. Některé významné soubory a adresáře jsou popsány v tabulce B.1. Samotný souborový systém `/proc` je podrobněji popsán na manuálové stránce `proc`.

Souborový systém `/var` obsahuje data podléhající změnám, například logovací soubory, dočasné soubory apod.

V tabulce C.1 jsou uvedeny některé důležité soubory a adresáře souborového systému `/var`. Tabulky B.1 a C.1 jsou převzaty z [10].

1.4.2 Nástroje pro monitorování hardwaru

Nástrojů pro zobrazování informací o aktuálním stavu počítače je velké množství. Většinou se jedná menší o programy spouštěné z příkazového řádku, jejichž výstupem bývá textová informace zobrazující stav sledované veličiny. Podobu výstupu jde obvykle měnit pomocí tzv. přepínačů.

Monitorování hardwaru představuje monitorování na nejnižší úrovni. Monitorovat stav hardwaru je užitečné a dá se tak v čas odhalit „umírající“ hardware.

Lm-sensors

Lm-sensors je sada nástrojů, které slouží k monitorování hardwaru na Linuxových systémech. Umí monitorovat teploty, napětí a otáčky ventilátorů. Hlavní součástí lm-sensors jsou ovladače jednotlivých monitorovacích obvodů a čipových sad, které se vyskytují na základních deskách. Obsahuje také skript `sensors-detect` pro detekci senzorů. Pro zobrazení sledovaných informací slouží příkaz `$ senzors`. Více informací o lm-sensors je možné najít na [12].

Hddtemp

Jak už název napovídá, jedná se o malý prográmek zobrazující teplotu pevného disku. Hodnoty získává čtením Monitoring Analysis and Reporting Technology (S.M.A.R.T.). Nástroj může být spouštěn z příkazové řádky příkazem `$ hddtemp /dev/disk` (disk je označení pevné disku v systému, např. `sda`), nebo běžet jako démon na pozadí. Více informací o tomto nástroji je možné najít v jeho manuálových stránkách.

Smartmontools

Smartmontools je balíček, který obsahuje dva nástroje (`smartctl` a `smartd`) pro monitorování pevných disků pomocí technologie S.M.A.R.T.. v GNU/Linuxu. Výpis Monitoring Analysis and Reporting Technology (S.M.A.R.T.) informací je možné provést zadáním příkazu `& smartctl -a /dev/disk`, kde `disk` představuje označení disku v systému (např. `sda`). Další informace o tomto nástroji a o technologii S.M.A.R.T. jsou dostupné na [4].

1.4.3 Nástroje pro monitorování zátěže systému

Monitorování zátěže systému může vést k odhalení chybně nakonfigurovaných služeb, které mohou příliš zatěžovat procesor, spotřebovávat velké množství paměti apod.

Top

Nástroj top je součástí balíčku procps. Top zobrazuje komfortní statistiku systému – informace o systému (doba běhu, počet přihlášených uživatelů, průměrná zátěž v různých časových intervalech), statistiku procesů (počet aktivních a běžících procesů), využití paměti, procesoru a swapu a v neposlední řadě také seznam procesů. Kromě toho všeho umí pracovat s procesy (zabíjet je, nastavovat jim různou prioritu) a mnohé další. Top se spouští jednoduše zadáním příkazu `$ top`. Více informací je možné najít v manuálových stránkách programu top.

Free, swapon

Pro pouhé vypsání statistiky obsazení paměti a odkládacího prostoru je k dispozici příkaz free, který je také součástí balíčku procps. Pro detailní statistiku obsazení jednotlivých odkládacích oddílů (swapu) je možné použít příkaz `$ swapon -s`. Samotný příkaz `$ swapon` jinak slouží k přidávání nebo odebírání odkládacích oddílů. Podrobnější informace je možné najít v manuálových stránkách.

Vmstat

Vmstat periodicky vypisuje informace o počtu běžících či zablokovaných procesů, stavu paměti, stránkování, I/O operacích blokových zařízení, počtu přerušení, počtu přepnutí kontextu a vytížení procesoru. Je rovněž součástí balíčku procps. Vmstat čte informace z příslušných souborů v `/proc`. Je možné zadat 2 parametry, a to prodlevu mezi jednotlivými výpisy a počet výpisů. První řádek výpisu uvádí průměrné hodnoty sledovaných parametrů od restartu systému (s výjimkou statistiky procesů a paměti, ta je vždy aktuální). Vysvětlení zobrazovaných parametrů a další informace o tomto nástroji jsou dostupné na [9].

Dstat

Dstat představuje víceúčelový monitorovací program nahrazující nástroje jako vmstat, iostat, netstat, nfsstat, ifstat. Dstat periodicky vypisuje údaje o zátěži, paměti, swapu, procesech, přerušeních, zátěži procesoru, času, stránkování, síti a dalších zdrojích. Pomocí přepínačů lze nastavit zobrazování každého z údajů, stejně tak perioda statistiky nebo počet vypsaných řádků. Dstat tedy pohodlně shrnuje mnoho různých monitorovacích programů, převyšuje některé jejich možnosti a navíc poskytuje kontinuální přísun aktuálních informací. Více informací je možné najít na [26].

Sysstat

Balíček Sysstat obsahuje několik výkonných monitorovacích nástrojů – sar, sadf, iostat, nfsiostat, cifsiostat, mpstat, a pidstat. Poskytuje celkový pohled na aktuální výkon systému. Hlavním účelem je zobrazit uživateli všechny důležité parametry najednou a dát obecnou představu o tom, co se děje v systému. Umožňuje také export dat do různých formátů (CVS, XML atd.) pro archivaci nebo pro grafické zpracovaní. Více informací o tomto nástroji je možné najít na [9] a [8].

1.4.4 Nástroje pro pokročilejší monitorování

Pokročilejší monitorovací nástroje umí nejen to co výše zmíněné, ale navíc umožňují monitorovat i konkrétní služby a získaná data ukládat a tvořit z nich například webové stránky s grafickými přehledy. Také mohou disponovat GUI pro „snadnější“ konfiguraci.

Protokol SNMP

Protokol SMNP (Simple Network Management Protokol) je asynchronní, transakčně orientovaný protokol založený na modelu klient-server pro správu síťových prvků. Strana, která posílá požadavky (snmp klient), může být např. jednoduchý snmp browser či složitý NMS (Network Management System), na straně zařízení je snmp agent (snmp server), který na požadavky odpovídá. Výjimku tvoří tzv. trapy, které agenti vysílají asynchronně při výskytu jednotlivých událostí (výpadek proudu, větráku, překročení mezních údajů, objevení nového zařízení). Pro přenos dat se používá protokol UDP [17]. Typickými úlohami pro SMNP protokol jsou:

- zjišťování stavových informací o zařízeních (množství volné paměti, počet spojení, počet přihlášených uživatelů apod.),
- nastavování parametrů (atributů) na síťových prvcích,
- monitorování uptimu,
- monitorování verzí běžících systémů,
- sběr dat o existujících síťových rozhraních(ifName, ifDescr, ifSpeed, ifType, ifPhysAddr),
- measuring network interface throughput (ifInOctets, ifOutOctets),
- dotazování ARP (ipNetToMedia).

V současné době existují tři verze protokolu SNMP (1, 2 a 3). SNMPv2 má navíc oproti SNMPv1 implementovánu autentizaci. SNMPv3 je doplněna o šifrování. Zařízení obvykle podporují všechny tři verze, ale v dnešní době už jsou SNMPv1 a SNMPv2 považovány za zastaralé.

SNMP je pouze protokol pro přenos informací. Pro jejich zpracování je nutné použít aplikaci, která tento protokol podporuje.

MRTG

Multi Router Traffic Grapher (MRTG) je software pro monitorování síťových zařízení pomocí SNMP protokolu. MRTG pracuje tak, že se periodicky v určeném časovém intervalu (defaultně každých 5 minut) dotazuje monitorovaných zařízení na jejich stav a získává od nich informace o velikosti síťového provozu, zatížení CPU nebo využití operační paměti. Z nasbíraných dat poté generuje HTML stránky

s grafy. MRTG je naprogramován v Perlu, a proto je dostupný jak na Linux, Unix, Windows, Mac OS tak i na NetWare od Novellu. MRTG je vyvíjen pod GPL licencí. Informace o používání toho systému je možné získat na [7].

Nagios

Nagios je robustní dohledové centrum, které slouží pro proaktivní monitorování a správu síťových zařízení (servery, pracovní stanice, diskové pole, síťové prvky, tiskárny, atd.), jejich stavu a služeb na nich provozovaných včetně operačních systémů, databází, pošty, webových stránek a jiných systémových nebo uživatelských aplikací.

V případě výpadku monitorovaného zařízení nebo služby systém dokáže poslat prostřednictvím mailu nebo GSM upozornění správcům sítě nebo samostatně provést některou z předem definovaných akcí - restart problémové služby apod. Monitorování je prováděno několika způsoby (např. pomocí ping, nebo SNMP). Systém je možno rozšiřovat prostřednictvím uživatelských funkcí (pluginů) a lze tak monitorovat velké množství zařízení a služeb, definovat vlastní monitorovací postupy a reakce na různé stavy. Nagios obsahuje webové rozhraní, kde je zobrazován stav kontrolovaných zařízení, upozornění a historie událostí.

Systém byl navržen na Linuxu jako Open Source. Monitorovat lze prakticky vše včetně počítačů s operačními systémy Linux, UNIX, Windows, přes telefonní ústřednu až po fyzikální veličiny jako je teplota, tlak apod.

Více informací o tomto produktu je možné získat na [14].

Munin

Munin je jednou z nejznámějších monitorovacích aplikací, která se používá především na serverových stanicích. Je napsaný v jazyku Perl a je dostupný pro většinu platforem. Munin shromažďuje různá systémová data jako například vytížení procesoru, síťový trafik, obsazenost diskových oddílů, využití paměti atd. Výstupem je webová stránky s grafy, které jsou periodicky aktualizovány.

Aplikace Munin se skládá ze dvou částí, které také odpovídají balíčkům. První komponentou je aplikace munin master, která představuje serverovou část, tzv. uv-grafový server. Ta se stará o poskytování výstupních dat, tj. grafů, které se ukládají ve formě obrázků do zvoleného adresáře.

Druhou komponentou je aplikace munin node, která představuje uzel pro přístup grafového serveru. Jejím úkolem je zpřístupnění stanice grafovému serveru.

Zdrojová data může Munin získávat z lokální stanice nebo také ze vzdálených serverů. Navíc existuje pro Munin velké množství pluginů pro monitorování množství parametrů různých aplikací, například pluginy pro web server, ftp server, poštovní

server, atd. Některé pluginy jsou však použitelné jenom na některých platformách (Linux, FreeBSD, NetBSD, Solaris, AIX, HP-UX, atd.). Munin také umožňuje vkládání vlastních pluginů [3].

Logwatch

Je nástroj pro monitorování a filtrování systémového logu. Je schopný provést jeho kompletní audit a analýzu, informovat o aktivitě serveru, neautorizovaných přístupech a stavu jednotlivých běžících služeb systému. Takto zpracované informace je pak možné odeslat na e-mail.

Další aplikace

Na poli open source software lze ještě zmínit aplikace Cacti, Zenoss, Splunk, v komerční zóně Tivoli, HP Openview, GroundWork.

2 NÁVRH VLASTNÍ APLIKACE

Jak ukazuje přehled v sekci 1.4.4, aplikací pro monitorování systému existuje celá řada. Proč tedy vytvářet vlastní aplikaci? Důvodů je několik. Hlavním je, že ani jedna aplikace zcela nepokrývá stanovené požadavky a bylo by nutné zvolit jejich kombinaci. Dále pak je to složitost konfigurace u rozsáhlých systému (jako např. Nagios) a velké časové nároky na získání znalostí k efektivní obsluze aplikace.

Úkolem je tedy navrhnout aplikaci, která zastane všechny požadované úkoly, bude snadno rozšiřitelná a konfigurovatelná a pro svůj chod bude spotřebovávat co nejméně systémových prostředků.

2.1 Specifikace požadavků

Před samotným návrhem je vhodné stanovit požadavky, které jsou na aplikaci kladeny. Ty lze shrnout do následujících bodů:

- kompatibilita s platformou GNU/Linux,
- schopnost monitorovat a zaznamenávat různé veličiny (služby) na jedné nebo více stanicích (serverech) umístěných v počítačové síti,
- zpracování získaných dat do přehledné formy a jejich poskytnutí uživateli,
- snadná rozšiřitelnost o další služby,
- zabezpečení proti vniknutí neoprávněné osoby.

2.2 Obecný návrh

Činnost celého systému je možné rozdělit do čtyř hlavních úloh.

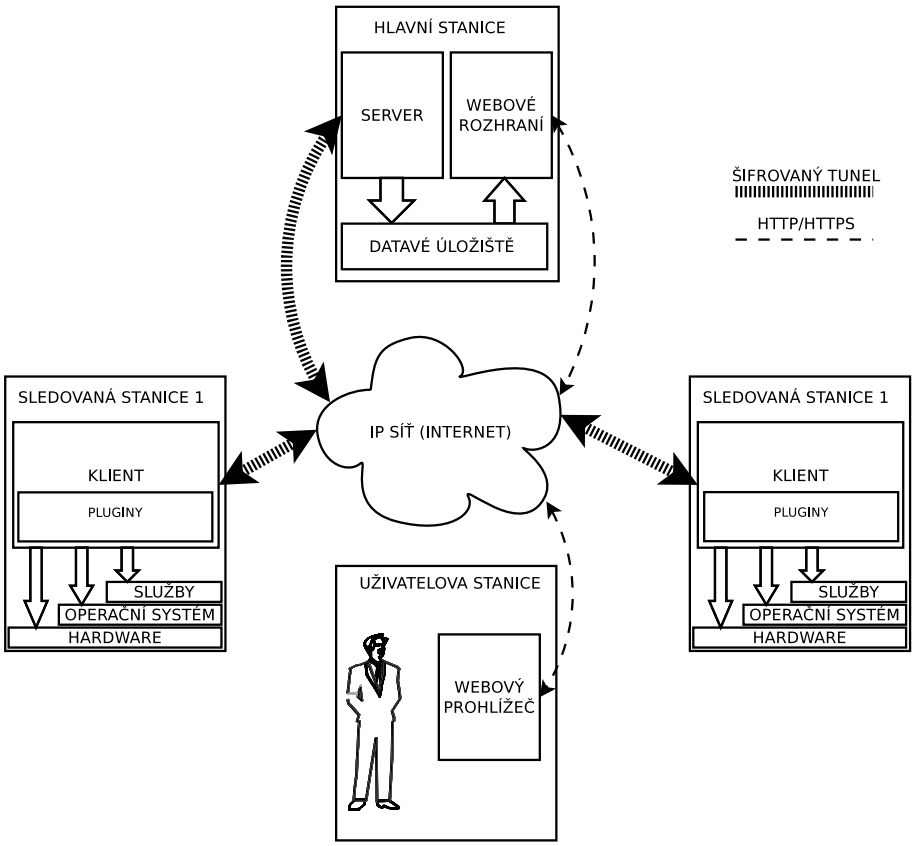
1. Získání požadovaných dat.
2. Zpracování a uložení dat.
3. Uchování dat.
4. Zobrazení dat uživateli.

Dále je možné celý systém rozdělit na několik autonomních modulů, kdy každý modul bude vykonávat jednu ze čtyř výše uvedených úloh.

1. klient,
2. server,
3. datové úložiště,
4. webové rozhraní.

2.2.1 Stručný popis funkce

První dva úkoly bude obstarávat aplikace typu klient-server. Úkolem klienta bude shromažďování požadovaných dat z monitorované stanice a odesílaní těchto dat v pravidelném intervalu přes počítačovou síť serverové části, která se bude fyzicky nacházet na jiné stanici. K tomuto účelu bude využit Přenášená data budou zašifrována pomocí SSL vrstvy. Serverová část bude tato data třídit a vkládat do datového úložiště. Poslední část aplikace bude přistupovat k datovému úložišti a z uložených dat generovat webové stránky s grafickými přehledy a informacemi o monitorované stanici. Pro přístup k webovému rozhraní bude vyžadována autentizace uživatele. Obrázek 2.1 zjednodušeně ilustruje činnost všech částí aplikace.



Obr. 2.1: Zjednošuné schéma činnosti celé aplikace.

2.3 Výběr technologií pro implementaci aplikace

Aplikaci, tak jak byla popsána, lze realizovat několika způsoby. V následující kapitole jsou popsány technologie zvolené pro jednotlivé části aplikace. Technologie byly voleny především s ohledem na osobní zkušenosti, stav dokumentace a v neposlední

řadě kompatibilitu s operačním systémem GNU/Linux. Důraz při výběru byl také kladen na to, aby zvolené produkty byly distribuovány jako Open Source.

2.3.1 Klient a server

Pro implementaci klienta a serveru se nabízí množství různých programovacích jazyků, ať už kompilovaných či interpretovaných. Kompilované jazyky jsou obvykle rychlejší, ale programování může být pracnější. Programování v interpretovaných jazycích může být snadnější, ovšem cenou za toto pohodlí je výkon, který je nižší než kompilovaných jazyků.

Jako programovací jazyk pro implementaci klienta a serveru byl zvolení jazyk C++ a to především z těchto důvodů:

- osobní zkušenost,
- příbuznost s jazykem C,
- standardní knihovna s řadou užitečných tříd a rozhraní,
- objektově orientovaný přístup pro snadnější rozšíření a přehlednost,
- kompatibilita se systémem GNU/Linux.

Jazyk C++

Jazyk C++ vychází z jazyka C, jehož autory jsou Ken Thompson a Dennis Ritchie. C++ navrhl Bjarne Stroustrup v Bellových laboratoří, jenž na jazyce začal pracovat na počátku osmdesátých let. Původně byl jazyk C++ vyvíjen pro vyřešení jedné konkrétní simulace s velice specifickými požadavky, které napovídaly spíše na využití jiného jazyka než C. Do roku 1983 přidal Stroustrup další vlastnosti do jazyka C a vzniklo tak něco, co on sám nazval „C with Classes“ (C s třídami). Roku 1983 nový jazyk poprvé opustil prostory Bellových laboratoří a na svou pouť světem se vydal pod označením C++, které bylo tehdy prvně použito [15].

Stroustrup založil C++ na stručnosti C, přidal rysy OOP aniž by významně změnil složku C. C++ je tedy nadstavba znamenající, že každý program napsaný v C je také platným programem pro C++. Programy v C++ mohou využívat softwarové knihovny C.

2.3.2 Datové úložiště

Z charakteru aplikace vyplývá, že bude nutné ukládat různé typy dat. Jednak informace o sledovaných stanicích a dále pak statistická data pro tvorbu grafických přehledů. Dalším požadavkem na datové úložiště je rychlý přístup k jednotlivým položkám. Vhodným datovým úložištěm bude tedy databáze.

Pro uchovávání statistických dat se často využívá cyklická databáze RRD, její největší výhodou je, že po vytvoření má stále stejnou velikost a nenarůstá. To je způsobeno tím, že po vyčerpání volného místa se začnou přepisovat starší záznamy. Spolu s RRDTool pak tvoří mocný nástroj pro sběr statistických dat a tvorbu grafů. V tomto případě však bude potřeba uchovávat nejen statistická data (čísla), ale i data textového charakteru. Aby byla data pohromadě bude vhodnější zvolit relační databázi.

Při výběru databáze bylo přihlédnuto k několika aspektům:

- rychlost databáze,
- osobní zkušenost,
- podpora ze stran zvolených programovacích jazyků,
- možnost práce s daty přímo v databázi (pomocí triggerů a procedur),

Nejlépe tyto požadavky splnil databázový systém MySQL.

MySQL

MySQL je relační databázový systém typu DBMS (database managment system), který vychází z deklarativního programovacího jazyka SQL (Structured Query Language). MySQL byla vytvořena švédskou firmou MySQL AB a je šířena jako open source. Jedná se o malý, rychlý a jednoduchý databázový systém. MySQL nabízí několik typů tabulek. Každý typ má své výhody a nevýhody a hodí se pro různé situace. Jedním z hlavních rozdílů je podpora transakcí[1].

Systém MySQL disponuje některými specifickými vlastnostmi, díky kterým se stal rozšířenější než konkurenční databázové systémy, mezi něž patří například Oracle, PostgreSQL, SQLite, Firebird.

Vlastnosti MySQL

- Nejdůležitější vlastností databáze je její stabilita. MySQL je velmi stabilní. Každá nová verze je vždy vývojáři důkladně otestována.
- Další velmi důležitá vlastnost je rychlost. MySQL je téměř ve všech kategoriích nejrychlejší.
- MySQL je portována na většinu operačních systémů.
- MySQL podporuje přístup z mnoha programovacích jazyků (C, C++, Eiffel, Java, Perl, PHP, Python a Tcl).
- Poměrně dynamický vývoj MySQL.
- MySQL je standardně zdarma.

[1]Transakce je uspořádaná skupina databázových operací (dotazů, procedur), která se vnímá a provádí jako jediná jednotka, a to celá, nebo vůbec ne. Nikdy nesmí nastat případ, kdy se vykoná jen její část.

- MySQL je dnes velmi rozšířená. Obrovská výhoda vyplývající z rozšířenosti je uživatelská podpora.

2.3.3 Webové rozhraní

Pro prezentaci dat uživateli se nabízí několik řešení. Je možné vytvořit aplikaci, která by se napojila na databázi a data ve vhodné (grafické) podobě předkládala uživateli, avšak je zde omezení, že uživatel by musel být při prohlížení záznamů fyzicky přítomen u stanice.

Elegantnější řešení představuje prezentovat sledovaná data prostřednictvím webové stránky. Hlavní výhodou tohoto řešení je, že webová stránka je prakticky dostupná odkudkoliv, tudíž odpadá nutnost fyzické přítomnosti u stanice. Nevýhodou je nutnost použití odlišného programovacího jazyka, jelikož C++ není primárně určen pro vytváření webových stránek (aplikací).

Jako nejvhodnější volba pro tvorbu dynamických webových stránek se jeví použití některého ze skriptovacích jazyků (jako např. Perl, PHP, Python apod.). Pro implementaci webového rozhraní byl na základě následujících kritérií zvolen jazyk PHP.

- rozšířenost,
- dobrá podpora databáze MySQL,
- jednoduchost.

Jazyk PHP

PHP je serverový skriptovací jazyk (server-side) navržený pro potřeby webových stránek. To znamená, že vše co PHP provádí neprobíhá na straně klienta jako například u JavaScriptu, ale interpretuje se na straně serveru a generuje HTML (či jiný) výstup, který vidí uživatel. PHP je Open Source, tedy volně šiřitelná technologie. PHP není závislé na platformě a není vázané s žádným konkrétním serverem, může tedy běžet kdekoli.

Počátky vývoje PHP sahají do roku 1994, kdy Rasmus Lerdorf[2] vytvořil jednoduchý systém v jazyce Perl pro zpracování záznamů o přístupech k jeho webu. Později byl systém přepsán do jazyka v C, jelikož perlovský kód značně zatěžoval server[linuxsof PHP]. Systém se rozšířil mezi další uživatele, kteří přicházeli s požadavky na vylepšení. Vznikl tak systém Personal Home Page Tools, později Personal Home Page Construction Kit.

[2]Rasmus Lerdorf (* 22. listopadu 1968 Qeqertarsuaq, Grónsko) je dánsko-kanadský programátor, který vytvořil první verzi webově orientovaného programovacího jazyka PHP.

Od svého vytvoření urazilo PHP dlouho cestu. V současné době je distribuováno již ve verzi 5, která podporuje OOP, disponuje snazší obsluhou chyb. Také je zde použit nový engine Zend Engine 2.0 [21].

2.3.4 Tvorba grafů

Statistická data je nejlepší prezentovat formou grafů. Grafy jsou přehledné a jde z nich rychle vyčíst v jakém stavu se sledovaná veličina nachází. Grafy je možné vytvářet několika způsoby, avšak nejvýhodnější by bylo, aby se grafy tvořily až při požadavku na jejich zobrazení, tedy v příslušném PHP skriptu. Tím by se odlehčilo serveru (snížila by se zátěž), jelikož by nebyly tvořeny neustále všechny grafy, ale pouze ty, které by chtěl uživatel vidět. Generování obrázkových grafů v samotném PHP je však velmi pracné [18]. Proto je vhodné použít nástroj pro tvorbu grafů přímo určený.

Velmi rozšířeným programem pro tvorbu grafů je Gnuplot, jenž je přenositelný mezi mnoha architekturami a lze jej používat jednak jako interaktivní nástroj, nebo jako automatickou utilitu pracující ve skriptu. Dále se nabízí použití knihovny napsané přímo v PHP (např. JPGraph, libchart, pchart apod.). S ohledem na použití PHP pro tvorbu webového rozhraní, kvalitu dokumentace a konfigurační možnosti, byla pro tvorbu grafů zvolena knihovna JPGraph.

Knihovna JpGraph

JPGraph je přídavná knihovna, která není standardní součástí distribuce PHP. Je šířena v komerční i nekomerční verzi, v nekomerční verzi pod licencí QPL (Qt Public License). Je napsaná v prostředí UNIXu, ale odzkoušená a plně funkční též pod Windows. Je plně objektově orientovaná. Její použití je snadné a vyžaduje pouze základní znalosti programování v PHP [18].

Pomocí JpGraph je možné kreslit, resp. generovat následující grafy:

- sloupcové,
- bodové,
- linkové,
- plošné,
- 3D,
- koláčové,
- pavučinové,
- čárové a QR kódy.

Výše jmenované typy grafů lze vykreslovat v mnoha podobách, různě kombinovat a podobně. Více informací o knihovně JPgraph včetně příkladů použití je množné najít na [1].

3 IMPLEMENTACE

Možností, jak pomocí zvolených technologií aplikaci realizovat, opět existuje celá řada. V této kapitole je popsáno, jakým způsobem jsou jednotlivé části aplikace navrženy a implementovány.

3.1 Klient

Klient představuje program, jenž je spuštěn na sledované stanici. Jeho úkolem je v pravidelném časovém intervalu provádět následující soubor činností:

1. získat aktuální hodnoty sledovaných veličin,
2. upravit tyto hodnoty do tvaru srozumitelného pro server,
3. odeslat data serveru.

Aby byl splněn požadavek na snadnou rozšiřitelnost, bude klient implementován modulárně, to znamená, že bude tvořen jádrem a pluginy[1].

3.1.1 Soubory obsahující zdrojové kódy klienta

`apss-client.cpp` – hlavní program klientské části (jádro).

`apss-client.conf` – konfigurační soubor klientské části.

`client.h` – hlavičkový soubor třídy `client`. Tato třída tvoří základ klientského programu, obsahuje metody pro načtení zásuvných modulů, metody pro práci se soketovým[2] API, které je použito pro komunikaci se serverem a další pomocné metody.

`client.cpp` – tento soubor obsahuje zdrojové kódy metod třídy `client`.

`except.h` – třída sloužící k ošetření výjimek.

`plugin.h` – soubor obsahuje abstraktní třídu, jež definuje rozhraní pro pluginy, každý plugin je odvozen z této třídy.

`ConfigFile.h` – hlavičkový soubor třídy `ConfigFile`, třída se používá k načítání konfiguračních dat z konfiguračního souboru. Zdrojové kódy byly staženy z [25].

`ConfigFile.cpp` – implementace třídy `ConfigFile`.

`Triplet.h` – z této třídy dědí třída `ConfigFile`.

`log.h` – hlavičkový soubor třídy `log`, tato třída je využívána pro přidávání zpráv do logovacího[3] souboru. Původní zdrojové kódy byly staženy z [22].

[1] Zásuvný modul neboli plugin, také plug-in je software, který nepracuje samostatně, ale jako doplňkový modul jiné aplikace a rozšiřuje tak její funkčnost.

[2] Soket je mechanizmus pro komunikaci. Poprvé se objevil v operačním systému BSD. Soket je velice obecný nástroj. Stejné funkce můžeme používat pro komunikaci pomocí různých protokolů.

[3] Log nebo také žurnál je název pro záznam nebo soubor záznamů (často soubory s příponou log), které si některé programy vytvářejí pro ukládání informací o své činnosti a běhu. Logy slouží

Před použitím byly mírně upraveny.

`log.cpp` – implementace třídy `log`.

`Makefile` – soubor obsahuje specifikaci pravidel pro překlad programu pomocí utility make[4].

3.1.2 Stručný popis činnosti klienta

Činnost klientského programu ilustruje vývojový diagram na obrázku 3.1. Po spuštění programu dojde nejprve k inicializaci, při níž jsou uživatelem nastavitelné parametry načteny z konfiguračního souboru. Následně jsou načteny zásuvné moduly (načítaní modulů je podrobněji popsáno v části 3.1.3), které mají být spuštěny.

Proběhne-li vše v pořádku, program vstoupí do hlavní programové smyčky a v cyklu spouští načtené zásuvné moduly. Každý modul získá hodnotu své sledované veličiny. Po shromáždění dat od zásuvných modulů proběhne komunikace se serverem, při níž se data odešlou serveru. Poté program čeká zadaný časový interval. Po jeho vypršení se soubor operací v hlavní programové smyčce opakuje.

3.1.3 Načtení pluginů

Načítání pluginů zajišťuje metoda `LoadPlugin`, která je implementována ve třídě `client`. Metoda využívá rozhraní `dlopen.h`, to je napsáno v jazyce C a tudíž je také primárně určeno pro načítání pluginů napsaných v jazyce C.

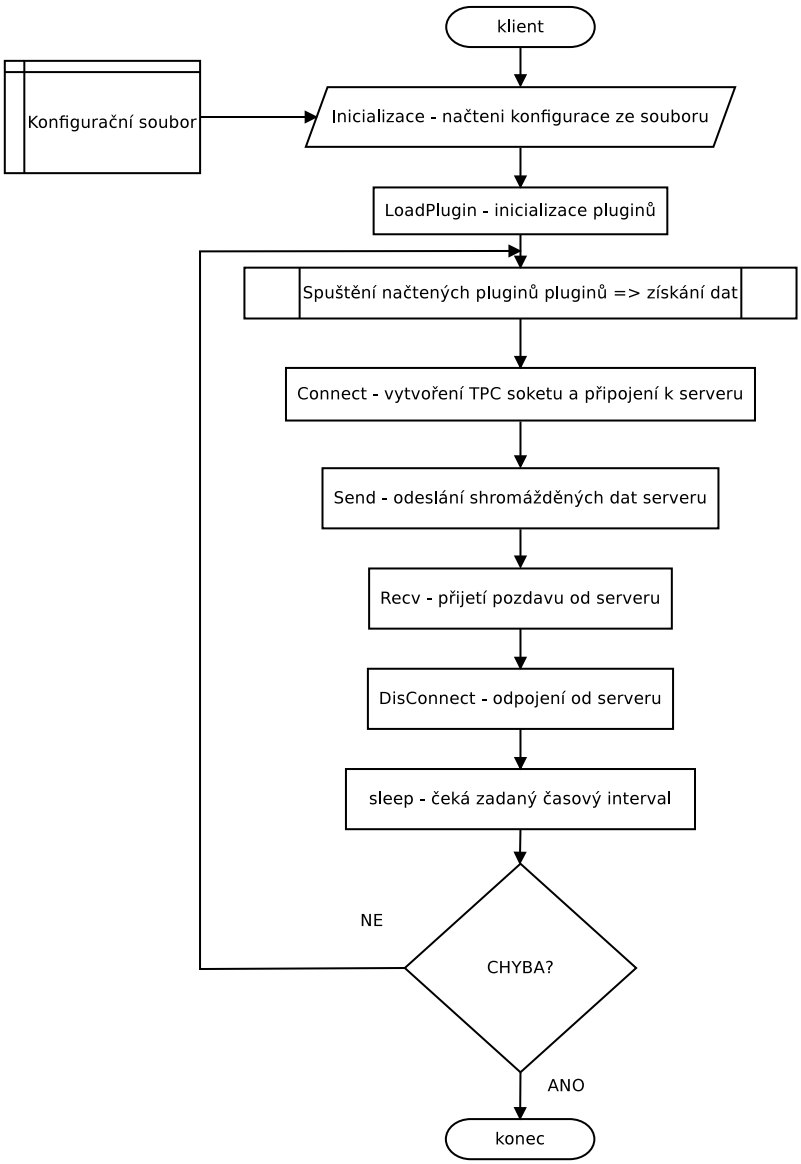
Existuje však i způsob, kterým lze pomocí `dlsym.h` načíst pluginy napsané v jazyce C++. Je nutné vytvořit rozhraní (abstraktní třída `plugin`) s čistě virtuálními členy a umístit jej do hlavního programu (jádra). Každý plugin je pak od tohoto rozhraní odvozen. Kromě čistě virtuálních členů obsahuje rozhraní ještě dvě speciální funkce (tzv. *class factory functions*) deklarované jako `extern "C"`. První vytváří instanci třídy a vrací ukazatel na ni, druhá přebírá ukazatel vytvořený první funkcí a ničí instanci dané třídy. Podrobnější informace o této problematice lze nalézt v [11].

Funkce `LoadPlugin` přebírá jako jeden z parametrů odkaz na vektor `modules` a naplní ho strukturami obsahujícími veškerá potřebná data pro práci s pluginem (ukazatelem na instanci pluginu apod.). Počet prvků ve vektoru určuje počet načtených pluginů.

---

při zpětné analýze k rozpoznání, zda došlo k nějaké chybě a pakliže ano, pak pomáhají určit, k jaké chybě došlo a proč.

[4]Make je program pro automatické kompilování programů, které jsou složeny z více souborů.



Obr. 3.1: Zjednodušený vývojový diagram popisující činnost klienta.

3.1.4 Spuštění pluginů

Spuštění pluginu představuje zavolání metod `setPlugin`, `collectData`, `formatData`, `getData` na objekt typu `plugin`. Prakticky to znamená, že se v cyklu prochází vektor struktur `modules` a na každý načtený plugin se zavolají výše zmíněné metody. Pluginy jsou podrobněji popsány v části 3.2.

3.1.5 Komunikace se serverem

Komunikaci se serverem zajišťují metody `Connect`, `Send`, `Recv`, `Disconnect` třídy `client`, které k tomu využívají soketové API jazyka C. Přenos dat probíhá po-

mocí protokolu TCP, který zajišťuje spolehlivou, spojově orientovanou službu. TCP soket pracuje v tzv. blokovacím režimu. To znamená, že soketová funkce `recv` zablokuje chod programu, dokud nepřijdou nějaká data. Teoreticky by tuto funkci klient nemusel vůbec používat, protože jeho činnost není závislá na žádných datech posílaných serverem (server posílá pouze pozdrav po přijetí dat), avšak pro lepší kontrolu průběhu spojení je využita.

Popis funkce jednotlivých metod

`Connect` – nejprve se vytvoří pomocí soketové funkce `socket`. Je-li soket úspěšně vytvořen, může dojít k navázání spojení se serverem. Aby však klient věděl, kam se má připojit, je nejprve nutné získat strukturu `sockaddr_in` [5]. Tato struktura přesně charakterizuje, kam má být spojení směrováno. Obsahuje jak IP adresu cílového počítače, tak i číslo portu, na kterém naslouchá serverová aplikace. Po úspěšném získání struktury `sockaddr_in` je možné navázat spojení se serverem. K tomu slouží funkce `connect`, která je rovněž součástí metody `Connect`.

`Send` – metoda přebírá jako parametr odkaz na řetězec, který obsahuje data od všech spuštěných pluginů, a tento řetězec odesílá pomocí funkce soketové `send` serveru.

`Recv` – metoda obsahuje soketovou funkci `recv`, přijme příchozí data do přijímacího bufferu, dále pak jsou data z bufferu zkopírována do členské proměnné, ze které mohou být získány pomocí metody `GetRecv`.

`DisConnect` – tato metoda obsahuje soketovou funkci `close`, pomocí níž dojde k uzavření spojení se serverem.

3.2 Pluginy

Pluginy jsou stěžejní částí celé aplikace. Každý plugin představuje sběrač dat pro jednu konkrétní službu (veličinu). Jelikož klientský program sám o sobě žádná data nezískává, je potřeba pro správnou funkci celého systému spustit alespoň jeden plugin. Aplikace podporuje dva typy pluginů:

1. textový – data získaná tímto typem pluginu jsou textového charakteru (např. přihlášení uživatelé, spuštěné procesy apod.). Textový plugin je charakterizován nulou (0).
2. číselný – pluginem získaná data představují číselnou hodnotu (např. teplota procesoru, otáčky větráčku apod). Číselný plugin je charakterizován jedničkou (1).

Typ pluginu určuje jakým způsobem budou data dále zpracována a jakou formou budou poskytnuta uživateli (textová v podobě výpisu a číselná v podobě grafu).

3.2.1 Soubory obsahující zdrojové kódy pluginu

`plugin.h` – soubor obsahuje abstraktní třídu, jež definuje rozhraní pro pluginy, každý plugin je z této třídy odvozen.

`plugin.cpp` – třída obsahující zdrojový kód pluginu.

`pluginFunctions.h` – tento hlavičkový soubor obsahuje prototypy funkcí, které je možné využít při tvorbě vlastním pluginů, případně je možné sem vkládat vlastní funkce.

`pluginFunctions.cpp` – soubor obsahuje implementace funkcí deklarovaných v souboru `pluginFunctions.h`, oba tyto soubory nejsou povinné.

3.2.2 Formát dat

Výstupem každého pluginu je řetězec, který obsahuje jméno pluginu, hodnotu získané veličiny a typ pluginu. Přičemž za každým atributem následuje středník (;). Pro plugin sledující teplotu procesoru by mohl výstup vypadat následovně:

```
1 cputemp;45;1;
```

Toto formátování zajišťuje metoda `formatData`. Pomocí metody `getData` je pak možné tento zformátovaný řetězec získat.

Jelikož je pravděpodobné, že bude spouštěno více pluginů, oddělují se jednotlivé výstupy uvozovkou (’). Výsledný řetězec vytvořený spuštěním několika modulů, může mít následující podobu:

```
1 cputemp;45;1;'cpuload;11.1;1;'loggedusers;petr pavel tonda;0;'
```

Před odesláním jsou data od pluginů ještě doplněná o hlavičku, ve které je přenášena IP adresa a jméno stanice.

```
1 192.168.10.22;muj_pocitac;'cputemp;45;1;'cpuload;11.1;1;'loggedusers;petr pavel
    tonda;0;'
```

Data v této podobě je pak už možné předat metodě `Send` k odeslání na server. Velikost odesílaných dat je závislá na přijímacím bufferu serveru, ten je nastaven na 10 000 bytů.

Při vytváření vlastního pluginu je nutné ošetřit, aby se žádný z oddělovacích znaků nevyskytl v jeho výstupu.

3.2.3 Získání dat

K získání dat slouží metoda `collectData`, v abstraktní třídě `plugin` je deklarována jako čistě virtuální a je nutné ji v odvozené třídě překrýt. Plugin může získávat data dvěma způsoby:

1. Použitím funkce `popen`, pomocí které je získán výstup nějaké systémové utility (např. některého z nástrojů uvedených v sekci 1.4.2 a 1.4.3).
2. Přečtením z příslušných souborů v souborovém systému `/proc` B.1 nebo `/var` C.1), případně přečtením z jiných souborů.

První způsob umožňuje každému uživateli poměrně snadné a rychlé vytvoření vlastního modulu. Uživatel předá funkci `popen` příkaz ve stejné podobě, v jaké by ho zadával do příkazové řádky svého shellu, funkce tento příkaz provede a vrátí jeho výstup.

Použití funkce `popen` má však několik nevýhod, na které je potřeba brát zřetel. Funkce `popen` pracuje se standardním vstupem a výstupem (`stdin` a `stdout`), jejichž deskriptory[5] (stejně jako deskriptor standardního chybového výstupu – `stderr`) se obvykle při běhu programu na pozadí z bezpečnostních důvodů uzavírají. Aby tedy bylo možné funkci `popen` použít při běhu programu na pozadí, je nutné nechat tyto deskriptory otevřeny. Je-li tedy v konfiguračním souboru nastaven běh klientského programu na pozadí jsou deskriptory `stdin` a `stdout` otevřeny.

Další nevýhodou jsou vyšší nároky na systémové prostředky, jelikož po zavolaní funkce `popen` je spuštěn shell jako nový proces a v něm je příkaz vykonán. Více informací o této funkci a práci s ní je možné nají v [19].

Vytvoření pluginu druhým způsobem je pracnější a jsou kladeny vyšší nároky na znalosti uživatele. Uživatel musí vědět odkud a jakým způsobem požadovanou informaci získat, je při tom zcela odkázán na možnosti jazyka C++. Tento způsob lze ale považovat za bezpečnější, jelikož je možné při běhu na pozadí uzavřít všechny tři deskriptory (`stdin`, `stdout` i `stderr`) a méně náročnější z hlediska systémových zdrojů.

3.2.4 Vytvoření vlastního pluginu

Vytvoření vlastního pluginu spočívá v tom, že se vytvoří nová třída odvozená ze třídy `plugin` a překryjí se čistě virtuální metody `setPlugin` a `collectDATA`. Výpis 3.1 ukazuje zdrojový kód pluginu, který získává vytížení procesoru v procentech.

```
1 #include <string>
2 #include <algorithm>
3 #include "pluginFunctions.h"
```

[5] Souborový deskriptor je abstraktní klíč, který unixové operační systémy používají pro přístup k souboru.

```
4  #include "plugin.h"
5
6  using std::string;
7
8  class cpuload : public plugin{
9          public:
10                 int collectData();
11                 void setPlugin();
12 };
13
14 void cpuload::setPlugin(){
15         pluginName = "cpuload";
16         pluginType = "0";
17 }
18
19 int cpuload::collectData(){
20         data = getStdoutFromCommand("top -n 2 | awk '/Cpu/{print $5}' | cut -d '%'
               -f 1 | tail -n 1 | awk '{print 100-$1}'");
21         replace(data.begin(), data.end(), ';', ' ');
22         replace(data.begin(), data.end(), '\'', ' ');
23
24         return 0;
25 }
26
27 // the class factories
28 extern "C" plugin* create(){
29         return new cpuload;
30 }
31
32 extern "C" void destroy(plugin* p){
33         delete p;
34 }
```

Výpis 3.1: Příklad pluginu pro sledování vytížení procesoru v procentech.

Řádky 1 až 4 zahrnují inkluzi potřebných hlavičkových souborů. Na řádcích 8 až 13 je deklarována třída `cpuload`, která dědí z třídy `plugin`.

Na řádcích 15 až 19 dochází k překrytí metody `setPlugin`. V této metodě se nastavují řetězce jméno pluginu `pluginName` a typ pluginu `pluginType`, Řetězec `pluginName` musí být unikátní, protože identifikuje plugin. Na tuto skutečnost je třeba brát ohled při tvorbě více pluginů.

Metoda `collectDATA` je překryta na řádcích 21 až 28. Tato metoda je určena k získání hodnoty dané veličiny. Hodnota se ukládá do členské proměnné `data`. V případě číselného pluginu je to jedna číselná hodnota (např. `12`, `18.56`, `1552` apod.), v případě textového pluginu libovolný text. Je čistě na uživateli jak tuto metodu implementuje, důležité je pouze zajistit, aby byl výsledek uložen to proměnné `data`. Ve výpisu 3.1 je na řádku 23 hodnota získána zavoláním funkce `getStdoutFromCommand`, které přebírá jako parametr shellovský příkaz a vrací výstup tohoto příkazu. Funkce `getStdoutFromCommand` je deklarována v knihovně `pluginFunctions` a používá funkci `popen`. Na řádcích 24 a 25 uveden příklad ošetření výskytu oddělovacích znaků. Znaky jsou nahrazeny mezerou.

Na řádcích 31 až 39 jsou deklarovány speciální funkce (tzv. *class factory functions*), zde je potřeba na řádku 39 zadat typ objektu korespondující s názvem třídy.

Kompilace a instalace pluginu

Plugin musí být zkompilován jako sdílená knihovna (přípona `.so`), to zajistí přepínač `-shared`. Kompilaci vzorového pluginu z výpisu 3.1 lze provést například následujícím příkazem:

```
1 gcc -Wall -shared -o cpu_load.so pluginFunctions.cpp cpu_load.cpp
```

Instalace se provede tak, že se výstupní soubor, v tomto případě `cpu_load.so`, nakopíruje do adresáře, ve kterém jádro klientského programu očekává pluginy. Pro spuštění pluginu je nutné přidat jeho jméno do parametru `Plugins` v konfiguračním souboru `apss-client.conf`.

3.3 Server

Serverová část aplikace je program, jehož úkolem je naslouchat na přiděleném portu a vyčkávat na žádost o připojení od klienta. Po navázání TCP spojení server přijme data od klienta a tyto data roztřídí a uloží do databáze. Z charakteru aplikace vyplývá, že server bude přijímat spojení maximálně od desítek klientů, kteří se navíc budou připojovat periodicky po relativně dlouhém intervalu (řádově desítky sekund) a pouze na krátký okamžik, při kterém dojde k obsluze klienta (řádově setiny sekundy). Z tohoto důvodu je server realizován jako jednovláknový a jednoprocesový a v jeden okamžik je schopen obsloužit pouze jednoho klienta, požadavky na spojení od ostatních klientů jsou řazeny do fronty.

Jelikož C++ nemá ve své implementaci podporu databází, je nutné pro práci s databází použít externí knihovnu. Volba padla na MySQL++, což je C++ API pro práci s databází MySQL.

3.3.1 Soubory obsahující zdrojové kódy serveru

`apss-server.cpp` – hlavní program serveru.

`apss-server.conf` – konfigurační soubor serveru.

`server.h` – hlavičkový soubor třídy `server`. Tato třída tvoří základ serveru, obsahuje metody pro zpracování dat přijatých od klienta, metody pro práci se soketovým API, které je použito pro komunikaci s klientem a další pomocné metody.

`server.cpp` – tento soubor obsahuje zdrojové kódy metod třídy `server`.

`except.h` – třída slouží k ošetření výjimek.

`ConfigFile.h` – hlavičkový soubor třídy `ConfigFile`, třída se používá k načítání konfiguračních dat z konfiguračního souboru. Zdrojové kódy byly staženy z [25].

`ConfigFile.cpp` – implementace třídy `ConfigFile`.

`Triplet.h` – z této třídy dědí třída `ConfigFile`.

`log.h` – hlavičkový soubor třídy `log`, tato třída je využívána pro přidávání zpráv do logovacího souboru. Původní zdrojové kódy byly staženy z [22]. Před použitím byly mírně upraveny.

`log.cpp` – implementace třídy `log`.

`Makefile` – soubor obsahuje specifikaci pravidel pro překlad programu pomocí utility make.

3.3.2 Stručný popis činnosti serveru

Činnost serveru ilustruje vývojový diagram na obrázku 3.2. Start serveru probíhá obdobně jako u klienta. Nejprve dochází k načtení parametrů z konfiguračního souboru a vytvoření potřebných objektů.

Poté je vytvořen soket a fronta požadavků na spojení. Server naslouchá na přiděleném portu a čeká na připojení klienta. Program nyní přejde do hlavní programové smyčky. Jakmile se ve frontě objeví požadavek na připojení od klienta, je akceptován a proběhne kontrola (na základě údajů v konfiguračním souboru), zda-li připojený klient má povolení se serverem komunikovat. Pokud ne, je okamžitě odpojen. Pokud ano, server příjme data od klienta, odešle mu pozdrav a ukončí spojení.

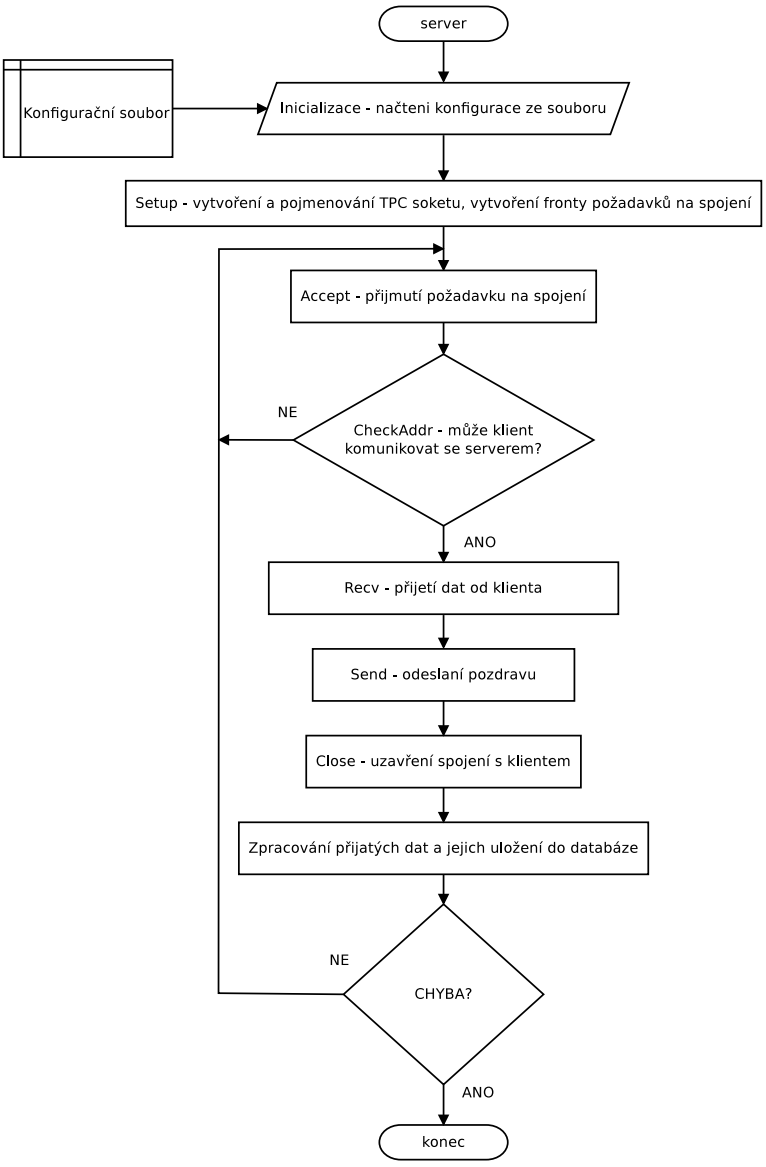
Následuje zpracování přijatých dat a jejich uložení do databáze. Tímto je klient obsloužen a server může akceptovat další požadavek na připojení z fronty. Pokud je fronta prázdná, čeká na připojení klienta a činnost programu je zablokována až do příchodu dalšího požadavku na spojení.

3.3.3 Komunikace s klientem

Jak už bylo řečeno, komunikace mezi klientem a serverem probíhá pomocí TCP soketů. Soketové funkce pro server jsou mírně odlišné než funkce pro klienta. Komunikaci s klientem zajišťují metody implementované ve třídě `server`.

Popis funkce jednotlivých metod

`Setup` – tato metoda vytvoří TCP soket pomocí soketové funkce `socket`, dále tento soket pojmenuje pomocí soketové funkce `bind` a nakonec funkcí `listen` vytvoří frontu požadavků na spojení. Velikost fronty je 20 požadavků, při naplnění fronty jsou další požadavky odmítány.



Obr. 3.2: Vývojový diagram popisující činnost serveru.

`Accept` – metoda pomocí soketové funkce `accept` vybere požadavek na spojení z fronty a potvrdí ho.

`Recv` – metoda obsahuje soketovou funkci `recv`, přijme příchozí data do přijímacího bufferu, dále pak jsou data z bufferu zkopírována do členské proměnné, ze které mohou být získány pomocí metody `GetRecv`. Velikost přijímacího bufferu je nastavena 10 000 B.

`Send` – metoda odesílá pozdrav pomocí soketové funkce `send` klientovi.

`Close` – tato metoda obsahuje soketovou funkci `close`, pomocí níž dojde k uzavření spojení se klientem.

3.3.4 Zpracovaní přijatých dat

Přijatá data představují řetězec přesně v takové podobě v jaké byl odeslán (tedy za předpokladu že komunikace nebyla odposlechnuta a data modifikována) z klienta. Při zpracovaní řetězce jsou v podstatě provedeny inverzní operace, než které vedly k jeho sestavení. Řetězec je tedy rozdělen na podřetězce, které jsou ohraničeny oddělovacími znaky a ty jsou pak vloženy do databáze.

Dělení řetězce zajišťuje metoda `SplitString`, která vstupní řetězec rozdělí podle zadaného oddělovacího znaku na podřetězce a ty uloží do vektoru. Dělení probíhá na dvou úrovních.

Nejprve je řetězec rozdělen na podřetězce oddělené uvozovkou (’). První získaný podřetězec obsahuje hlavičku a dále se zpracovává samostatně, ostatní pak data od každého pluginu spuštěného na klientovi.

V druhé úrovni se v cyklu prochází vektor s uloženými podřetězci, znovu dochází k rozdělení pomocí metody `SplitString`, tentokrát je oddělovacím znakem středník (;), získá se tak další vektor řetězců obsahující jméno pluginu, hodnota sledované veličiny a typ pluginu. Server následně provede dva databázové dotazy, kterými vloží data do příslušných tabulek v databázi.

Jak jsou data ukládaná do databáze je podrobněji popsáno v části 3.6.

3.4 Logovaní

Obě části aplikace klient-server jsou schopny podávat informace o své činnosti zápisem zpráv do logovacího souboru. Tato činnost musí být nejprve povolena v konfiguračním souboru. Logovány jsou jednak různé chyby, ke kterým může dojít například při navazování spojení, komunikaci, práci s databází apod., dále programy logují průběh spojení. Může tak být snadno zkontrolováno, zda-li jsou odesílaná data v požadovaném tvaru například při vytváření nových pluginů. Po odladění je možné logovaní vypnout, což může ušetřit systémové prostředky. Logovaní zajišťuje třída `log`. Ukázku logu klienta ilustruje výpis 3.2, log serveru je zobrazen ve výpisu 3.3.

```
1  (04/16/2011 20:11:28): Startuji apss-client...
2  (04/16/2011 20:11:28): Nacitam plugin: /usr/lib/apss/cpuload.so
3  (04/16/2011 20:11:28): Nacitam plugin: /usr/lib/apss/cputemp.so
4  (04/16/2011 20:11:32): cpuload;7;0;
5  (04/16/2011 20:11:32): cputemp;42.5;0;
6  (04/16/2011 20:11:32): CHYBA: Nelze navazat spojeni: Connection refused
7  (04/16/2011 20:11:32): Ukoncuji apss-client...
8  (04/16/2011 20:12:07): Startuji apss-client...
9  (04/16/2011 20:12:07): Nacitam plugin: /usr/lib/apss/cpuload.so
10 (04/16/2011 20:12:07): Nacitam plugin: /usr/lib/apss/cputemp.so
11 (04/16/2011 20:12:11): cpuload;7.5;0;
12 (04/16/2011 20:12:11): cputemp;43.5;0;
```

```
13 (04/16/2011 20:12:11): Navazano spojeni se serverem.
14 (04/16/2011 20:12:11): Odeslano: 32 bytu.
15 (04/16/2011 20:12:11): 192.168.10.22;muj_pocitac;'cpuload;7.5;0;'cputemp;43.5;0;'
16 (04/16/2011 20:12:11): Prijato: 18 bytu.
17 (04/16/2011 20:12:11): Hello apps-client
18 (04/16/2011 20:12:11): Uzaviram spojeni se serverem.
```

Výpis 3.2: Výpis logu klientského programu.

```
1 (04/16/2011 21:43:42): Nekdo se pripojil z~adresy 127.0.0.1
2 (04/16/2011 21:43:42): Prijato: 30 bytu.
3 (04/16/2011 21:43:42): 192.168.10.22;muj_pocitac;'cpuload;9;0;'cputemp;43.5;0;'
4 (04/16/2011 21:43:42): Odeslano: 18 bytu.
5 (04/16/2011 21:43:42): Hello apps-client.
6 (04/16/2011 21:43:42): Uzaviram spojeni s~127.0.0.1
```

Výpis 3.3: Výpis logu serverového programu.

3.5 Zabezpečení

Klient i server jsou naprogramováni tak, že je může spustit pouze superuživatel root. Avšak běh aplikace s takto vysokými právy se obecně nepovažuje za bezpečný. Pokud by například útočník dokázal nějakým způsobem modifikovat zdrojový kód některého pluginu, nebo podstrčit uživateli plugin se zákeřným kódem, mohl by se dostat k důvěrným datům nebo systém jiným způsobem zneužít či poškodit.

Aby tato skutečnost byla omezena, je při instalaci vytvořen nový uživatel bez možnosti přihlášení a bez domovského adresáře a nová skupina a po spuštění programu (klienta i serveru) jsou práva pomocí funkcí `setuid` a `getuid` „snížena“ na tohoto uživatele a skupinu.

Bezpečnostní riziko spojené s použitím funkce `popen` v pluginech bylo rozebráno v části 3.2.3

Povolení komunikovat se serverem mají pouze klienti jejichž IP adresy jsou zapsány v parametru `AllowClients` v konfiguračním souboru serveru. Kontrolu provádí metoda `CheckAddr` a pokud IP adresa není v konfiguračním souboru uvedena, je klient odpojen a nedojde ke komunikaci. Kontrolu na základě IP adresy je možné provádět až po akceptování požadavku na spojení, neboť do té doby program nezná IP adresu klienta. Toto řešení sice zamezí nepovolenému klientovi (útočníkovi) odeslat data na server, ale nezabrání mu v tom, aby server zahltil požadavky na spojení (DoS útok) a tím znemožnil připojení legitimních klientů. Jako obrana může posloužit aktivace tzv. SYN cookies a správně nakonfigurovaný firewall.

Přenos dat mezi klientem a serverem probíhá prostřednictvím protokolu TCP. Tento protokol není nijak kryptograficky zabezpečen a je tedy možné přenášené

segmenty odposlechnout a pozměnit, případně jinak zneužít. V části 4.6.1 je popsáno jak přenos dat zabezpečit pomocí programu Stunnel[6] a protokolu SSL[7] .

3.6 Databáze

Databáze slouží jako hlavní úložiště dat. Data jsou vkládána do tabulek. MySQL podporuje několik různých typů tabulek (tzv. úložiště dat), přičemž každý typ má své výhody a nevýhody. Nejpoužívanější jsou typy MyISAM a InnoODB. Z charakteru aplikace se jeví jako výhodnější použít úložiště dat InnoDB, neboť lze očekávat, že vkládání dat bude převažovat před jejich vybíráním. InnoDB oproti typu MYISAM dále umožňuje tvorbu cizích klíčů a transakce [20].

Mimo ukládání dat je databáze schopna zastat i část aplikační logiky a pomocí uložených procedur provádět s daty definované operace. To je výhodné například z hlediska rychlosti, jelikož veškeré operace provádí přímo v databázi databázový server. S tím souvisí i vyšší bezpečnost – data nejsou přenášena mimo databázi. Další výhodou jsou různé funkce, které databáze nabízí. Uložené procedury jsou použity pro přepočítávání číselných hodnot na průměrné, tato problematika je popsána v části 3.6.2.

3.6.1 Tabulky

Na obrázku je zobrazen 3.3 je ER diagram zobrazující jednotlivé tabulky, jejich atributy a vztahy mezi nimi.

`clients` – do této tabulky se ukládá IP adresa a jméno stanice, které byly zadány v konfiguračním souboru klienta. Dále je zde uloženo jméno, typ pluginu, identifikátor, který jednoznačně určuje o jaký plugin na jakém klientovi se jedná, a čas připojení.

`number_values` – do tabulky jsou ukládány hodnoty všech číselných pluginů, z hodnot zde uložených se generují denní grafy.

`string_values` – tato tabulka slouží k ukládání hodnoty všech textových pluginů

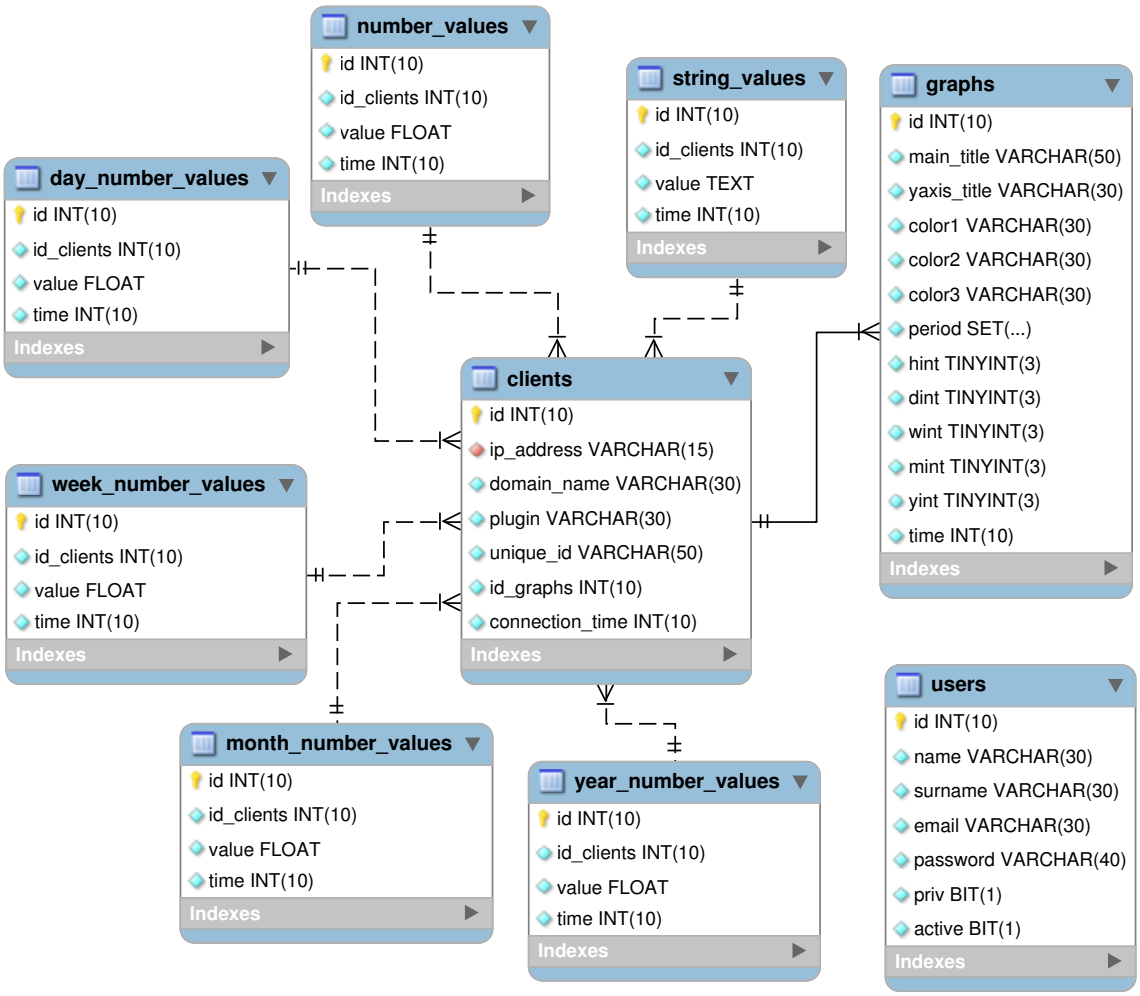
`*_number_values` – tyto tabulky jsou určeny k uložení přepočítaných číselných hodnot. První slovo v názvu uvádí, pro generovaní jakého grafu jsou data z těchto tabulek použita.

[6]Stunnel je software určený především jako doplněk k programům, které samy o sobě neumožňují šifrovat komunikaci.

[7]SSL a jeho novější verze TLS jsou vrstvy, které se vkládají mezi transportní (např. TCP) a aplikační vrstvu (např. HTTP) a zajišťují zabezpečení komunikace pomocí šifrování a autentizace.

`graphs` – údaje v této tabulce ovlivňují vzhled grafů, jsou zde uloženy popisky, barvy, typy zobrazovaných grafů a intervaly zobrazovaných hodnot. Všechny tyto parametry může uživatel nastavovat ve webovém. rozhraní.

`users` – v této tabulce uloženy informace o uživatelích, majících přístup do webového rozhraní. Je zde uloženo jméno, příjmení, e-mail – ten slouží jako login, heslo zahašované pomocí hašovací funkce SHA1, privilegia a stav účtu.



Obr. 3.3: ER diagram databáze.

3.6.2 Přepočítávání

Aby byla zajištěna zpětná kontrola i shromážděných hodnot a zároveň se zamezilo neustálému růstu databáze, je potřeba zvolit soubor hodnot a určit pro něj (například pomocí nějaké statistické funkce) jednu hodnotu, která jej bude dále reprezentovat. Tuto hodnotu uložit a ostatní hodnoty odstranit případně archivovat

mimo databázi. To zajistí jednak možnost zpětné kontroly starších hodnot a také relativně stálou velikost databáze. Toto řešení je však možné použít pouze u číselných hodnot. Hodnoty textových pluginů je nutné pravidelně odmazávat a pro případnou zpětnou kontrolu archivovat na jiném místě.

Statistických funkcí pro realizaci přepočítávání se nabízí celá řada, jako nejvhodnější se v tomto případě jeví požití mediánu[8], jeho hlavní výhodou je, že není ovlivněn extrémními hodnotami.

Funkce `median` není standardně v MySQL obsažena, existuje však UDF[9] funkce `median`, kterou je možné do databáze přidat. Zdrojový kód funkce včetně instalačního návodu je dostupná na stránkách [23].

Příklad uložené procedury, provádějící přepočet je uveden ve výpisu 3.4. Konkrétně se jedná o proceduru, která přepočítává hodnoty pro hodinové grafy z tabulky `number_values` na hodnoty pro denní grafy (tabulka `day_number_values`). Vstupním parametrem je časový úsek v sekundách, ten vymezuje množství dat, se kterými bude operace provedena. Procedura si uloží aktuální čas v podobě Unix timestamp[10], odečte od něj časový úsek předaný v parametru a vybere z tabulky `number_values` všechny záznamy, u kterých je čas (všechny časové údaje v databázi jsou ukládány jako Unix timestamp) větší než výsledná hodnota. Z vybraného souboru hodnot zjistí medián a jeho hodnotu spolu s časem, který byl uložen na začátku, vloží do tabulky `day_number_values`. Tento proces se opakuje pro všechny veličiny v tabulce `number_values`. Naprosto analogicky probíhá přepočet i u ostatních tabulek. Více informací o tvorbě uložených procedur je možné najít na [27].

```
1  DELIMITER //
2  CREATE PROCEDURE get_median_day(settime INT(11))
3      BEGIN
4
5      DECLARE tmp_id_clients INT(10);
6      DECLARE tmp_value FLOAT(3);
7      DECLARE s_id_clients INT(1);
8      DECLARE cur_time INT(11);
9      DECLARE done BIT(1) DEFAULT 0;
10
11     DECLARE client CURSOR FOR
12     SELECT id_clients FROM number_values GROUP BY id_clients HAVING (count(
           id_clients)>=1);
13
14     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

[8]Medián je hodnota, jež dělí řadu podle velikosti seřazených výsledků na dvě stejně početné poloviny. Ve statistice patří mezi míry centrální tendence. Platí, že nejméně 50 % hodnot je menších nebo rovných a nejméně 50 % hodnot je větších nebo rovných mediánu.

[9]Databáze MySQL nabízí uživatelům rozhraní pro vytváření vlastních funkcí, které nejsou standardně v systému obsaženy. Ty funkce jsou označovány jako User defined functions neboli uživatelsky definované funkce.

[10]Unix timestamp udává počet sekund od 1. 1. 1970 00:00:00 UTC. Používá se pro jednodušší ukládání data a času v jednom čísle.

```
15
16      SET cur_time = UNIX_TIMESTAMP();
17
18      OPEN client;
19      client_loop: LOOP
20      FETCH client INTO s_id_clients;
21      IF done THEN LEAVE client_loop; END IF;
22
23                  SELECT id_clients, median(value)
24                  INTO tmp_id_clients, tmp_value
25                  FROM number_values WHERE id_clients = s_id_clients AND time >= (
                        cur_time - settime) ;
26
27          INSERT INTO day_number_values (id_clients, value, time)
28      VALUES(s_id_clients, tmp_value, cur_time);
29
30      END LOOP client_loop;
31      CLOSE client;
32
33      END; //
34  DELIMITER ;
```

Výpis 3.4: Uložená procedura pro přepočet hodinových hodnot na denní.

Pro automatizování přepočítávaní je využit program `event scheduler`, jenž je podobný linuxovému `cronu`, s tím rozdílem, že je integrován přímo v databázi. Jedná se vlastně o plánovač, který podle zadaného rozvrhu automaticky vykonává požadované operace.

V tabulce 3.1 je uvedeno jakým způsobem jsou data přepočítávána. Je uvažován interval odesílání 1 minuta. Pokud by bylo nepřetržitě připojeno 10 klientů po dobu 10 let a každý by měl spuštěno 10 pluginů (uvažováno pouze číselných) byl by maximální počet záznamů v databázi přibližně 1 mil.

typ grafu	Hodinový graf	Denní graf	Týdenní graf	Měsíční graf	Roční graf
tabulka	`number_values`	`day_number_values`	`week_number_values`	`month_number_values`	`year_number_values`
hodnot pro přepočet	10	9	8	8	-
perioda přepočítávání	10 minut	90 minut	12 hodin	4 dny	-
perioda mazání	24 hodin	14 dní	12 týdnů	24 měsíců	10 let
záznamů k smazaní	1440	2016	1344	1344	840

Tab. 3.1: Přepočet číselných záznamů v databázi.

Promazávání a zálohování záznamů

Jak bylo již bylo řečeno v předchozí části, záznamy v databázi je nutné pravidelně mazat, aby nedocházelo k jejímu neustálému růstu. Promazávaní je řešeno pomocí plánovače, který v intervalech uvedených v tabulce 3.1 spouští příkaz `delete` nad každou promazávanou tabulkou. Mažou se vždy data shromážděná v předešlé periodě. Takže například u hodinového grafu se z tabulky `number_values` smažou data,

která byla zobrazována předcházející den. To zajišťuje, že tabulka i po promazání obsahuje dostatečné množství dat pro tvorbu grafů. Ve výpisu 3.5 je zobrazen SQL kód, jímž se vytvoří nová událost pro plánovač.

```
1 DELIMITER $$
2 CREATE EVENT `numberDeleter`
3 ON SCHEDULE EVERY 24 HOUR STARTS '2011-03-23 01:00:00'
4 DO
5 DELETE FROM `number_values` WHERE time <= (UNIX_TIMESTAMP() - 60*60*24)$$
```

Výpis 3.5: Vytvoření nové událost pro plánovač.

Mazaní údajů je nastaveno vždy na 1 hodinu po půlnoci, neboť přesně o půlnoci bude každý den databáze pomocí bashovského skriptu, jenž bude spouštěn cronem, zálohována.

3.7 Webové rozhraní

Webové rozhraní zajišťuje interpretaci hodnot uložených v databázi v podobě webových stránek. Jelikož je jeden z požadavků zamezit přístupu do systému neoprávněné osobě, obsahuje i mechanismus ověření uživatele a jednoduchou správou uživatelských účtů. Jako základ pro vzhled webového rozhraní posloužila šablona stažená ze stránek [24]. Ta obsahuje statický HTML a CSS kód, který byl podle potřeb upraven a doplněn PHP a JS[11] kódem.

3.7.1 Adresářová struktura

Webová aplikace je tvořena sadou několika souborů rozčleněných do adresářové struktury. V kořenu adresářové struktury se nalézají soubory `index.php` (Stánka s přihlašovacím formulářem), `apss.php` (hlavní skript), `css.css` (zdrojový soubor kaskádových stylů) a následující adresáře:

`app` – tento adresář obsahuje skripty zpracovávající data z uživatelských formulářů, soubor s funkcemi a skript pro generování grafů.

`db` – zde uložené soubory se využívají při práci s databází.

`img` – v tomto adresáři jsou uloženy všechny obrázky vyskytující se na stránkách, kromě obrázků s grafy.

`login` – adresář obsahuje skripty zajištující bezpečnost aplikace, například skript zpracovávající přihlašovací formulář, skript kontrolující oprávnění uživatele při prohlížení stránek apod.

`pages` – skripty v tomto adresáři jsou podle požadavků uživatele vkládány do v hlavního skriptu a generují jednotlivé stránky aplikace. Jsou zde například skripty

[11] JavaScript je programovací jazyk, který se používá při tvorbě internetových stránek.

generující konfigurační formuláře, skripty zajišťující zobrazení grafů či textového výpisu apod.

`src` – v tomto adresáři jsou zdrojové kódy javascriptu, které jsou použity pro realizaci rozbalovacího menu.

3.7.2 Prezentace textových dat

Textová data jsou prezentována v podobě dvousloupcového tabulkového výpisu. Levý sloupec obsahuje časový údaj, odpovídající času vložení hodnoty do databáze, pravý pak samotný text. Ještě než dojde k vložení textových hodnot do tabulky jsou předány funkci `txt2html`, která převede prostý text na HTML kód. Zobrazení textových dat zajišťuje skript `report.php`. Přímo na stránce s výpisem je pak možné změnit počet vypsaných záznamu.

3.7.3 Prezentace numerických dat

Numerické hodnoty jsou prezentovány formou grafů. Aplikace je schopna zobrazit až 5 grafů s různou časovou základnou. O zobrazení grafů se stará skript `graphloader.php`, který obdrží požadované grafy ve formě PNG[12] obrázků a vloží je na stránku.

3.7.4 Generování grafů

Grafy vytváří skript `graph.php`. Využívá k tomu knihovnu `Jpgraph`. Možnosti této knihovny jsou opravdu velmi široké a při realizaci aplikace byla využita jen jejich malá část. Skript je rozdělen na dvě části. V první je provedeno získání dat z databáze, ve druhé je z těchto dat vygenerován graf. Ve výpisu 3.6 je uvedena ta část kódu, která zajišťuje vykreslení grafu.

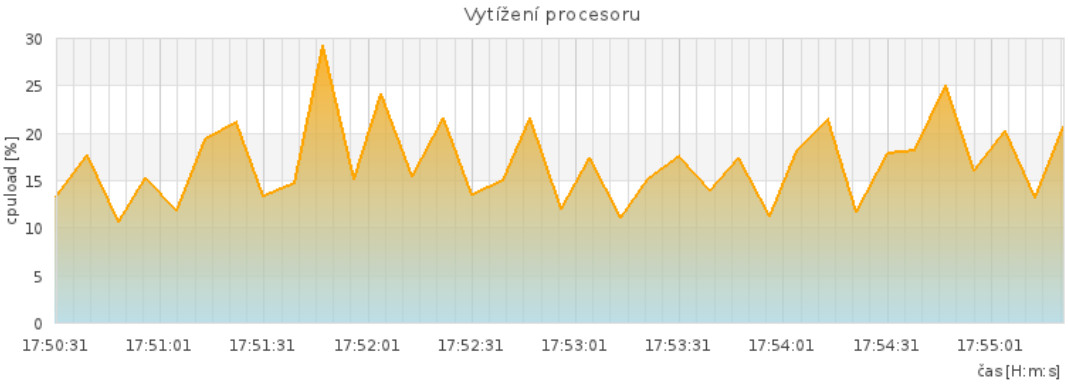
```
1  //Vytvoření instance grafu s~rozlišením 800x300 px
2  $graph = new Graph ( 850, 300 );
3  //Nadstavení okrajů
4  $graph->SetMargin ( 60, 20, 30, 50 );
5  //Nadstavení datového měřítka
6  $graph->SetScale ( "datlin" );
7  //Nastavení titulku grafu
8  $graph->title->Set ( $main_title );
9  //Nastavení osy X
10 $graph->xaxis->title->Set ( $xaxis_title );
11 $graph->xaxis->SetLabelAngle ( 0 );
12 $graph->xaxis->SetLabelFormatCallback ( $timecallback );
```

[12]PNG (Portable Network Graphics – přenosná síťová grafika) je grafický formát určený pro bezeztrátovou kompresi rastrové grafiky. Byl vyvinut jako zdokonalení a náhrada formátu GIF. PNG do jisté míry nahrazuje GIF, nabízí více barev, průhlednost a lepší kompresi. Nevýhodou PNG oproti GIF je praktická nedostupnost jednoduché animace.

```
13 $graph->xaxis->scale->ticks->Set ($interval*$main_tick, $interval*$tick);
14 $graph->xaxis->scale->SetTimeAlign ( $time_align );
15 $graph->xgrid->Show ( true, true );
16 //Nastavení osy Y
17 $graph->yaxis->scale->SetAutoMin ( 0 );
18 $graph->yaxis->title->Set ( $yaxis_title );
19 $graph->ygrid->Show ( );
20 $graph->yaxis->title->SetMargin(15);
21 //Vytvoření čárového grafu z předaných parametrů
22 $lineplot = new LinePlot ( $yvalues, $xtime );
23 //Nastavení barev přechodu
24 $lineplot->SetFillGradient ( $color1, $color2 );
25 //Nastavení anialiasingu a plynulých přechodů
26 $graph->img->SetAntiAliasing ( );
27 $graph->SetAlphaBlending ( );
28 //Přidání čárového grafu do instance grafu
29 $graph->Add ( $lineplot );
30 //Nastavení barvy čáry
31 $lineplot->SetColor ( $color3 );
32 //Vykreslení grafu
33 $graph->Stroke ( );
```

Výpis 3.6: Výpis PHP kódu zajištujícího vytvoření grafu.

Výstupem skriptu je obrázek ve formátu PNG, skript tak může být přímo předán jako zdroj obrázku v HTML tagu `img`, tedy například `<img src="graph.php">`. Příklad grafu vygenerovaného skriptem `graph.php`, je na obrázku 3.4. Všechny vygenerované grafy mají nezávisle proměnnou (osa X) pevně nastavenu na čas. Závisle proměnná (osa Y) je pak sledovaná veličina. Aplikace je tedy schopna zobrazovat pouze veličiny závislé na čase. Mřížku a hodnoty na ose X mění skript v závislosti na typu grafu velikosti zobrazovaného intervalu. Měřítko na ose Y se mění automaticky podle velikosti vstupních hodnot tak, aby plocha grafu bylo co nejvíce využita.



Obr. 3.4: Příklad grafu vygenerovaného skriptem `graph.php`

3.7.5 Konfigurace grafů

Změnit některé parametry grafu je možné přímo ve webovém rozhraní pomocí formuláře, který je generován skriptem `graphform.php`. Jde měnit titulek grafu, popisek osy Y, barvy gradientní výplně, barva čáry a zobrazovaný interval. Všechny tyto hodnoty se ukládají do tabulky `graphs` v databázi a jdou nastavit zvlášť pro každý plugin. Pokud uživatel vyžaduje změnu jiných parametrů, musí provést editaci ručně přímo ve skriptu `graph.php`, změny se pak budou týkat všech pluginů. Všechny možnosti, které knihovna JPGraph poskytuje, jsou podrobně popsány v dokumentaci dostupné na [1]

Mimo voleb, které přímo ovlivňují vzhled grafu, je možné přes formulář `graphform.php` také nastavit jaké typy grafů budou zobrazovány. Ovládaní webového rozhraní je podrobněji popsáno v části 5.

3.7.6 Uživatelské účty

Webová aplikace poskytuje jednoduchý mechanismus správy uživatelů účtů. Ty je možné přidávat a odebírat přes formulář generovaný skriptem `userform.php`. Po jeho vyplnění a odeslaní je do databázové tabulky `users` přidám nový záznam jednoznačně identifikující nového uživatele. Kromě jména, příjmení, e-mailu a zahašovaného hesla, obsahuje záznam také dva příznaky. První určuje typ účtu, a druhý je-li účet aktivní. V aplikaci mohou existovat dva typu uživatelských účtu, administrátorský a účet běžného uživatele:

administrátorský účet – uživatel s tímto účtem má v aplikaci neomezená práva a může měnit nastavení grafů a přidávat účty běžných uživatelů.

účet běžného uživatele – uživatel s tímto účtem může grafy pouze prohlížet, rovněž je mu znepřístupněn formulář pro správu uživatelů.

Po instalaci aplikace je automaticky vytvořen jeden administrátorský účet, který může spravovat účty běžných uživatelů. Po vytvoření nového účtu je účet označen jako neaktivní. Administrátor musí nějakým způsobem předat přihlašovací údaje běžnému uživateli, ten je při prvním přihlášení vyzván ke změně hesla. Po úspěšné změně hesla je účet aktivován a uživatel se může do aplikace přihlásit. Je-li potřeba více administrátorský účtů je nutné je přidat ručně vložením záznamu do tabulky `users`.

3.7.7 Zabezpečení

Jedním z hlavních požadavků na celou aplikaci je její zabezpečení proti vniknutí neoprávněné osoby. Zabezpečení webového rozhraní úzce souvisí s uživatelskými účty

popsanými v předchozí části. Před vstupem do webového rozhraní je provedena autentizace na základě znalosti přihlašovacích údajů (e-mail a heslo), pokud je nalezena shoda, dojde k autorizaci a uživatel má povolen vstup do webové rozhraní. Hesla jsou v databázi uloženy zhašované pomocí hašovací funkce SHA1[13] .

Přihlášení probíhá pomocí přihlašovacího formuláře na stránce `index.php`, skript `login_procesing.php` zpracovávající formulář ověří zadané údaje s údaji v databázi, pokud se udaje neshodují není přistup povolen. Pokud dojde ke shodě je vytvořena tzv. relace (session), pomocí které jsou udržovány informace o přihlášeném uživateli až do jeho odhlášení nebo vypršení časového limitu při nečinnosti. Při požadavku na zobrazení jakékoliv stránky webového rozhraní je pak pomocí session kontrolováno, zda-li požadavek pochází od přihlášeného uživatele. Pokud ne přístup na stránku není povolen.

Dalším prvkem, který zvýší bezpečnost aplikace je použíti protokolu HTTPS. Ten zajistí, že data přenášená mezi webovým prohlížečem uživatele a serverem budou zašifrovaná, tudíž nečitelná pro případného útočníka. Konfigurace podpory HTTPS na webserveru, je popsána v části 4.6.2.

[13]SHA (Secure Hash Algorithm) je rozšířená hašovací funkce, která vytváří ze vstupních dat výstup (otisk) fixní délky. Otisk je též označován jako miniatura, fingerprint, hash (česky někdy psán i jako haš). Jeho hlavní vlastností je, že malá změna na vstupu vede k velké změně na výstupu, tj. k vytvoření zásadně odlišného otisku.

4 INSTALACE A KONFIGURACE

Instalace a konfigurace aplikace je popsána pro operační systém Debian GNULinux. Analogicky by ji mělo být možné provést i na jiných linuxových distribucích. V tomto ukázkovém případě je pro zjednodušení server i klient nainstalován na jednu stanici.

Jednotlivé části aplikace jsou rozděleny do samostatných archivů. Před instalací je tedy nejprve nutné přenést, například pomocí programu SCP, potřebné archivy na stanici, na které bude probíhat instalace.

4.1 Instalace operačního systému

Debian GNU/Linux je možné zdarma stáhnout z jeho domovských stránek. Současná stabilní verze má číslo 6 a kódové značení „squeeze“. Podrobný návod na instalaci je rovněž dostupný na stránkách projektu.

Další postup uvažuje minimální systémovou instalaci pouze s nainstalovanými standardními systémovými nástroji. Po instalaci je vhodné obnovit databázi balíčků a případně provést aktualizaci. Například pomocí příkazů:

```
1 # apt-get update
2 # apt-get upgrade
```

Znak `#` na začátku řádku říká, že příkazy je nutné provést v příkazovém interpretu (shellu) superuživatele. Příkazový interpret běžného uživatele je uvozen znakem `$`.

4.2 Instalace potřebných knihoven a programů

Pro úspěšnou instalaci a běh aplikace je nutné doinstalovat několik knihoven a programů, které nejsou obsaženy v minimální instalaci. Tento software je ve většině případů dostupný přímo v repozitářích a jeho instalace je snadná, v ostatních případech budu nutné software stáhnout ručně a i ručně jej nainstalovat.

Dodatečný software je nutné instalovat hlavně pro serverovou část aplikace, pro instalaci klientské části postačí nainstalovat kompilátor a případně program Stunnel pokud je vyžadováno šifrování komunikace.

4.2.1 Software nutný pro kompilaci

Před instalací klienta a serveru je nejprve nutné tyto programy zkompilovat ze zdrojových souborů do binární spustitelného souboru. Pro úspěšné zkompilování je potřeba nainstalovat kompilátor. Například `g++` pomocí příkazu:

```
1 # apt-get install g++
```

Pokud se na stanici bude instalovat serverová část aplikace, je nutné nainstalovat ještě knihovnu `mysql++`.

```
# apt-get install libmysql++-dev libmysql++3 libmysqlclient-dev libmysqlclient16
    mysql-common zlib1g-dev
```

Nainstalovaní těchto balíčku by mělo zajistit bezproblémovou kompilaci aplikace.

4.2.2 MySQL

Dále je nutné nainstalovat databázi MySQL, například příkazem:

```
# apt-get install mysql-client mysql-server
```

Během instalace se objeví požadavek na zadání hesla pro superuživatele root.

4.2.3 Webový server

Pro zajištění chodu webového rozhraní a přístupu do něj je potřeba nainstalovat webový server. Webserverů existuje celá řada (např. Complex Web Server, Apache HTTP Server, Internet Information Services apod.). Mezi nejrozšířenější patří Apache HTTP Server, který bude použit i zde. Nainstalovat jej je možné pomocí příkazu:

```
# apt-get install apache2
```

Snadnější správu databáze MySQL je možné zajistit pomocí phpMyAdmin. Jedná se nástroj napsaný v jazyce PHP, který umožňuje spravovat databázi přes webové rozhraní. Instalaci je možné provést pomocí:

```
# apt-get install phpmyadmin
```

V průběhu instalace je možné zvolit automatickou konfiguraci pro webový server Apache. Dále pak ze zobrazí nabídka s možností automatického vytvoření databáze pro nastroj phpMyAdmin, při potvrzení této volby se následně zobrazí požadavek na zadání hesla superuživatele pro přístup do databáze.

4.2.4 Knihovna Jpgrah

Knihovna Jpgrah je v repozitářích Debianu pouze ve verzi 1.5, což je poslední verze vydaná pod licencí GPL. Aktuální verzi 3.5 je proto potřeba stáhnout ze stránek projektu. Instalace je pak poněkud náročnější a je popsána v dokumentaci[1]. Lze ji například provést pomocí následujících příkazů.

```
# mkdir /usr/share/php
# cd /usr/share/php
# wget http://jpgraph.net/download/download.php?p=5
# mv download.php\?p\=5 jpgraph.tar.gz
```

```
# tar -zxvf jpgraph.tar.gz
# ln -s /usr/share/php/jpgraph-3.5.0b1/src/ jpgraph
# rm jpgraph.tar.gz
```

Ve stabilní verzi (verze 6) operačního systému Debian se nachází zastaralá verze knihovny GD, kterou Jpgraph vyžívá k tvorbě grafů. Tento fakt způsobuje, že není možné využít funkce antialiasingu při generování grafu. Problém je možné odstranit ruční instalací aktuální knihovny GD nebo zakomentováním řádku způsobujícího chybové hlášení. Jedná se o řádek:

```
JpGraphError::RaiseL(25128);('The function imageantialias() is not available in
    your PHP installation. Use the GD version that comes with PHP and not the
    standalone version.')
```

Řádek se nachází v souboru `gd_image.inc.php`, jenž je umístěn v adresáři `/usr/share/php/jpgraphp/src`.

4.2.5 Stunnel

Stunnel není nezbytně nutný pro běh aplikace, zajišťuje však šifrování a dešifrování dat přenášených přes síť. Instalace se provede příkazem:

```
# apt-get install stunnel
```

Tímto je instalace potřebných komponent dokončena a je možné přejít k instalaci samotné aplikace.

4.3 Instalace a konfigurace klienta a serveru

Zdrojové kódy klienta a serveru jsou obsaženy v archivu `apss-client.tar.gz` respektive `apss-server.tar.gz`. Kromě zdrojových kódů klientského programu je v archivu `apss-client.tar.gz` také několik již zkompilovaných pluginů. V obou archivech se v adresáři `strartscript` nachází startovací skript. Před zahájením instalace je nejprve potřeba příslušný archiv rozbalit. Tedy například při instalaci klienta:

```
# tar -zxvf apss-client.tar.gz
```

Nyní je možné s využitím programu `make` program zkompilovat, pravidla pro překlad obsahuje soubor `Makefile`.

```
# make
```

Po úspěšné kompilaci je možné program nainstalovat opět pomocí `make` a přidáním cíle `install`.

```
# make install
```

Po spuštění instalace se nejprve vytvoří nový uživatel `apss` s UID 2222 (bez shellu a bez domovského adresáře) a nová skupina `apss` s GID rovněž 2222, s jejichž právy pak aplikace běží. Jedná se o bezpečnostní opatření zamezující spouštění příkazů v pluginech s právy superuživatele, podrobněji byla tato problematika popsána v částech 3.2.3 a 3.5.

4.3.1 Instalační adresáře

Všechny potřebné soubory se instalují do adresářů zadaných v souboru `Makefile`, změnit cílové adresáře je tedy možné přímou editací souboru `Makefile`. Ve výchozím nastavení jsou instalační cesty definovány následovně:

`/usr/bin` – do toho adresáře se instaluje spustitelný soubor `apss-client` respektive `apss-server`.

`/usr/lib/apss` – tento adresář je vytvořen pouze při instalaci klienta. Jsou do něj zkopírovány všechny moduly obsažené v archivu. Tento adresář je také nastaven v konfiguračním souboru `apss-client.conf` jako místo, odkud jsou načítány pluginy. Při vytváření vlastních pluginů je nutné přeložený plugin umístit do tohoto adresáře.

`/etc/apss` – tento adresář rovněž vytváří instalační skript, je do něj zkopírován konfigurační soubor `apss-client.conf` respektive `apss-server.conf`.

`/var/log` – v tomto adresáři je vytvořen v případě instalace klienta logovací soubor `apss-client.log`. V případě instalace serveru soubor `apss-client.log`.

4.3.2 Cíle pro make

Kromě `install` jsou definovány ještě cíle `all`, `uninstall`, `distrib` a `clean`.

`all` – přeloží program (stejné jako make).

`unistall` – smaže soubory vytvořené při instalaci.

`clean` – vymaže objektové soubory vzniklé po překladu.

`distrib` – vytvoří archiv ze zdrojových kódů.

4.3.3 Automatický start

Automatické spuštění programů lze zajistit vložením startovacích skriptů do adresáře `/etc/init.d/` a následným spuštěním příkazu `update-rc.d`[1] . Startovací skript navíc zajistí pohodlné a ovládání běhu programů pomocí `/etc/init.d/apss-client start|stop|restart`, podobně jako například u webového serveru apache. Skripty

[1] V instalaci na přiloženém DVD nebyl příkaz `update-rc.d` proveden a je tedy nutné programy spustit ručně.

se nacházejí v adresáři `startscript` v archivu příslušného programu. Postup uvedený níže je platný pro klienta.

```
# cd startscript
# cp apss-client /etc/init.d/
# update-rc.d apss-client default
```

Tímto by mělo být zajištěno automatické spuštění klienta po startu počítače. Provedené změny lze vrátit zpět pomocí příkazu:

```
# update-rc.d apss-client remove
```

4.3.4 Konfigurační soubory

Konfigurace klienta a serveru se provádí editací příslušného konfigurační souboru umístěného v adresáři `/etc/apss`.

Klient

Konfigurační soubor `apss-client.conf` obsahuje jednak parametry nutné pro správnou funkci programu, dále pak parametry, kterými je bezprostředně možné ovlivnit chovaní programu. Příklad, jakým způsobem může konfigurační soubor vypadat, je uveden ve výpisu 4.2.

`ServerAdress` – důležitý parametr, určuje adresu stanice, na které je spuštěna serverová část aplikace. Je možné zadat IP adresu nebo doménové jméno stanice.

`LocalAddress` – zde se zadává IP adresa stanice, používá se k identifikaci stanice.

`Name` – tento parametr určuje jméno, pod kterým bude stanice zobrazena ve webovém rozhraní.

`Port` – tento parametr určuje číslo portu, na kterém server naslouchá a očekává spojení.

`Interval` – určuje interval odesílání dat, to znamená dobu, po kterou program čeká před dalším spuštěním modulů a odesláním dat. Hodnota se zadává v sekundách. Výchozí hodnota je jedna minuta. Změnou této hodnoty je výrazně ovlivněna pouze podoba hodinového grafu, neboť tabulka `number_values`, z jejichž hodnot je graf generován, je plněna rychleji respektive pomaleji. To se projeví na rozlišení grafu: menší interval ⇒ více hodnot ⇒ podrobnější graf, případně: větší interval ⇒ méně hodnot ⇒ méně podrobný graf. Dále je také ovlivněn počet hodnot, ze kterých je určován medián a které jsou vkládány do tabulky `day_number_values`. To mírně ovlivní i vzhled denního grafu. V dalších grafech se již změny prakticky neprojeví.

`Deamon` – parametr určuje, zda-li se má program spustit jako démon na pozadí, či bude běžet na popředí.U klienta je to však spojeno s jistým rizikem, které bylo popsáno v části 3.2.3

`Logging` – pomocí tohoto parametru je možné zapnout či vypnout logovaní.

`LoggingFile` – parametr určuje cestu k logovacímu souboru

`Plugins` – zde se určí, jaké pluginy se mají spouštět. Uvádí se jména pluginů oddělená mezerou, jméno musí odpovídat názvu souboru pluginu bez přípony.

`PluginsPath` – parametr určuje cestu k adresáři, ve kterém jsou pluginy uloženy.

```
1  #adresa serveru
2  ServerAddress = localhost
3
4  #mistni IP adresa
5  LocalAddress = 192.168.10.237
6
7  #nazev ktery bude pouzit ve webovem rozhrani
8  Name = muj_pocitac
9
10 #port na kterem server server ocekava spojeni
11 Port = 12345
12
13 #perioda ziskavani a odesilani dat serveru
14 Interval = 5
15
16 #pragram se spusti na pozadi jako deamon
17 Deamon = true
18
19 #zapne/vypne logovani
20 Logging = true
21
22 #cesta k logovacimu souboru
23 LoggingFile =  /var/log/apss-client.log
24
25 #seznam pluginu ktere budou spousteny (za sebou oddelene mezerou)
26 Plugins = cpu_load cpu_temp
27
28 #cesta k adresari s pluginy
29 PluginsPath = /usr/lib/apss
```

Výpis 4.1: Konfigurační soubor `apss-server.conf`.

Server

Konfigurační soubor `apss-server.conf` podobně jako konfigurační soubor klienta obsahuje jednak parametry nutné pro správnou funkci programu, dále pak parametry, kterými je možné ovlivnit chování programu. Jeho možná podoba je zobrazena ve výpisu 4.2.

`Port` – tento parametr určuje číslo portu, na kterém bude server naslouchat a očekávat spojení.

`Database` – určuje jméno databáze se kterou aplikace pracuje.

`Dbserver` – adresa stanice, na které se databázový server nachází.

`Dbuser` – uživatelské jméno pro přístup do databáze.

`Dbpass` – heslo pro přístup do databáze.

`Logging` – pomocí tohoto parametru je možné zapnout či vypnout logovaní.

`LoggingFile` – parametr určuje cestu k logovacímu souboru.

`AllowClients` – zde se zadají IP adresy stanic, které mají povolené odeslání dat na server, vytváří se tak tzv. whitelist[2].

```
1 #urceni portu, na kterem serveru ocekava spojeni
2 Port = 12345
3
4 #pristupove udaje do databaze
5 #jmeno databaze
6 Database = apss
7 #jmeno adresa stanice s databazi
8 Dbserver = localhost
9 #uzivatel
10 Dbuser = apss
11 #heslo
12 Dbpass = apss
13
14 #zapne/vypne logovani
15 Logging = true
16
17 #program se spusti na pozadi jako demon
18 Deamon = true
19
20 #cesta k logovacimu souboru
21 LoggingFile = /var/log/apss-server.log
22
23 #adresy klietskych stanic kterym je povolena komunikace
24 AllowClients = 127.0.0.1
```

Výpis 4.2: Konfigurační soubor `apss-server.conf`.

4.4 Instalace a konfigurace databáze

Všechny soubory potřebné k vytvoření databáze jsou obsaženy v archivu `apss-databaze.tar.gz`. Je zde soubor `apss.sql`, jenž obsahuje strukturu celé databáze (tabulky, uložené procedury, události pro plánovač). Dále pak archiv `udf_median.tar.bz`, ten obsahuje zdrojový kód funkce `median`. A také je zde vložen skript `mysqlbackup.sh` pro zálohování databáze. Archiv lze rozbalit příkazem:

```
1 # tar -zxvf apss-databaze.tar.gz
```

Vytvoření databáze

Strukturu databáze je možné provést importem přes rozhraní phpMyAdmin nebo z příkazové řádky. Nejprve je ale nutné vytvořit databázi a uživatele. Nejprve tedy databázi:

[2]Whitelist (bílá listina) obsahuje subjekty, kterým je příslušná operace povolena.

```
# mysqladmin create <jméno> -u root -p
```

Parametr `<jméno>` reprezentuje název databáze (např. apss).

Pro vytvoření uživatele je již nutné přihlásit se k databázovému serveru.

```
# mysql -u root -p
```

Po úspěšném přihlášení se zobrazí příkazový interpret databázového serveru očekávající příkazy. Kromě samotného vytvoření uživatele je ještě potřeba mu přidělit databázi a pomocí práv specifikovat, jaké operace může s danou databází provádět. Pro zjednodušení budou uživateli přiřazena všechna práva. Případné změny je pak možné rychle provést pomocí phpMyAdmin.

```
> CREATE USER '<uživatel>'@'localhost' IDENTIFIED BY '<heslo>';
> GRANT ALL PRIVILEGES ON apss.* TO '<uživatel>'@'localhost' IDENTIFIED BY '<heslo>
    ';
```

Parametry `<uživatel>`, `<heslo>` představují login a heslo nového uživatel (obojí např. apss).

Nyní je databáze připravena a je možné do ní importovat její strukturu připravenou v souboru `apss.sql`. Předtím je však potřeba se odhlásit od databázového serveru.

```
# mysql <jméno> < apss.sql
```

Parametr `<jméno>` opět reprezentuje název databáze (např. apss). V tuto chvíli je instalace databáze téměř kompletní, zbývá ještě přidat funkci `median`.

4.4.1 Instalace funkce `median`

Jak už bylo řečeno v části 3.6.2 funkce `median` není standardně v databázi MySQL obsažena. Je proto nutné ji doinstalovat ručně. Nejprve je potřeba knihovnu zkompilovat a nakopírovat do adresáře, ve kterém ji databázový server najde. Instalační postup je následující:

```
# tar -zxvf udf_median.tar.bz
# g++ -o udf_median.so -shared -I/usr/include/mysql udf_median.cc
# cp udf_median.so /usr/lib/mysql/plugin/
```

Dále je nutné se přihlásit k databázovému serveru a zadat příkaz:

```
> CREATE AGGREGATE FUNCTION median RETURNS REAL SONAME 'udf_median.so';
```

Tímto je funkce `median` nainstalována a je možné ji používat stejným způsobem jako standardní funkce.

4.4.2 Plánovač

Plánovač událostí zajišťuje přepočítávání a promazávání záznamů v databázi. Ve výchozí instalaci databáze MySQL není plánovač události povolen, je proto nutné jej povolit dodatečně. Nejprve je potřeba se přihlásit k databázovému serveru:

```
# mysql -u root -p
```

Po zadání hesla superuživatele root je možné zadat příkaz, který zapne plánovač.

```
>  SET GLOBAL event_scheduler = ON;
```

Tímto je plánovač zapnut, ale jen do restartování databázového serveru. Pro automatické zapnutí po startu je potřeba vytvořit soubor `my.cnf` v adresáři `/etc`

```
# nano /etc/my.cnf
```

a vložit do něj následující řádky.

```
[mysqld]
event_scheduler=ON
```

4.4.3 Zálohování

Zálohování celé databáze bude probíhat každý den před promazáním tabulek, pomocí skriptu `mysqlbackup.sh`, který bude volán `cronem` každý den o půlnoci. Ve skript je nutné nejprve vyhledat a editovat několik řádků. Například takto:

```
MyUSER="apss"
MyPASS="apss"
MyHOST="localhost"

DEST="/var/backups/apss"

IGGY="mysql phpmyadmin information_schema"
```

Dále je nutné přidat dotabulky cronu (soubor `crontab`) řádek `0 0 * * * sh /<cesta>/mysqlbackup.sh`, který zajistí, že `cron` spustí skript každý den o půlnoci. Za `<cesta>` je samozřejmě nutné doplnit skutečnou cestu ke sriptu. Soubor `crontab` se otevře pro editaci například příkazem:

```
# crontab -e
```

Nakonec je ještě potřeba přidat skriptu spustitelný příznak a vytvořit adresář pro zálohy.

```
# chmod +x /<cesta>/mysqlbackup.sh
# mkdir /var/backups/apss
```

4.5 Instalace a konfigurace webového rozhraní

Adresářová struktura webového rozhraní včetně všech potřebných souborů je obsažena v archivu `apss-web.tar.gz`. Instalace spočívá v rozbalení archivu do adresáře, ve kterém webový server hledá soubory stránek, standardně je to adresář `/var/www`.

```
# cp apss-web.tar.gz /var/www
# cd /var/www
# tar -zxvf apss-web.tar.gz /var/www
# rm /var/www/apss-web.tar.gz
```

Jediné nastavení, které je potřeba udělat, je doplnit do souboru `db_connect` přihlašovací údaje do databáze. Editaci je možné provést pomocí příkazu:

```
# nano /var/www/db/db_connect.php
```

Výsledek by mohl vypadat takovýmto způsobem.

```
//--Vyplnit jméno databáze
$dbname = 'apss';
//--Vyplnit adresu db serveru
$host = 'localhost';
//--Uživatel databáze
$user = 'apss';
//--Heslo do databáze
$pass = 'apss';
```

V tuto chvíli je instalace aplikace kompletní a po zadání adresy `http://adresa-stanice/` do prohlížeče by se měl zobrazit přihlašovací formulář.

4.6 Zabezpečení

Zabezpečení aplikace proti vniknutí cizí osoby je jedním z hlavním požadavků. Nemá vliv na chod aplikace, ale zajišťuje důvěrnost dat jak při jejich přenosu přes síť po získání, tak i během kontroly ve webovém rozhraní. U přenosu dat přes síť je důvěrnost zajištěna pomocí programu Stunnel, který vytvoře mezi klientem a serverem šifrovaný SSL tunel. U webového rozhraním pak doplněním webserveru o podporu protokolu HTTPS.

4.6.1 Konfigurace programu Stunnel

Stunnel zajistí šifrování dat přenášených mezi klientem a servem pomocí SSL vrstvy. Nejprve je nutné na obou stanicích vygenerovat certifikát. K tomu lze využít balík openssl. Příkaz, kterým se vygeneruje certifikát s platností jednoho roku pak vypadá následovně:

```
# cd /etc/stunnel
# openssl req  -x509 -nodes -days 365  -newkey rsa:1024 -keyout stunnel.pem -out
    stunnel.pem
```

Během generování certifikátu je položeno několik otázek. Odpovědi na ně mohou vypadat například takto:

```
Country Name (2 letter code) [AU]:CZ
State or Province Name (full name) [Some-State]:Czech Republic
Locality Name (eg, city) []:Brno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:VUT
Organizational Unit Name (eg, section) []:FEKT
Common Name (eg, YOUR name) []:apss
Email Address []:apss@apss.cz
```

Poté je nutné editovat konfigurační soubor `stunnel.conf`, v něm zadat cestu ke vygenerovanému certifikátu a přidat definici služby, která má být tunelována. Ta se liší v závislosti na tom, je-li stanice server nebo klient. Na serveru tedy například:

```
cert = /etc/stunnel/stunnel.pem

[apss]
accept  = 12346
connect = 12345
```

Výše uvedené nastavení zajistí, že stunnel příjme šifrované TCP spojení na portu 12346, dešifruje jej a předá na port 12345, na kterém server očekává nešifrované spojení[3] . Tímto je možné komunikovat se serverem šifrovaně i nešifrovaně.

Na klientovi je nestavení obdobné, jen je potřeba odkomentovat parametr `client`, který určuje, že stanice se bude při komunikaci chovat jako klient.

```
cert = /etc/stunnel/stunnel.pem

client = yes

[apss]
accept  = 12345
connect = 192.168.10.235:12346
```

Všechny ostatní parametry je možné ponechat ve výchozím nastavení.

Dále je nutné v konfiguračním souboru klientské části `apss-client.conf` nastavit jako adresu serveru `localhost`.

```
ServerAddress = localhost
```

Tímto je dáno, že se klient nespojuje se serverem přímo, ale nejprve data předá programu stunnel, který očekává TCP spojení na portu 12345. Ten provede šifrování a naváže spojení se svým protějškem na serveru. Data jsou přeneseny šifrovaně.

K automatickému spuštění stunnelu po startu počítače je potřeba editovat jeho spouštěcí skript

```
# nano /etc/init.d/stunnel4
```

a v něm vyhledat a upravit parametr `ENABLED`.

[3]Tento port se definuje v konfiguračním souboru `apss-server.conf`

```
1 ENABLED=1
```

Dále je ještě nutné provést editaci v souboru `stunnel4`

```
1 # nano /etc/default/stunnel4
```

a upravit stejný parametr jako ve spouštěcím skriptu.

```
1 ENABLED=1
```

Komunikace pomocí stunnelu

Komunikace při použití stunnelu je znázorněna na obrázku 4.1. Klient a server spolu nekomunikují přímo, ale pouze lokálně předávají data stunnelu na přes rozhraní `localhost`.

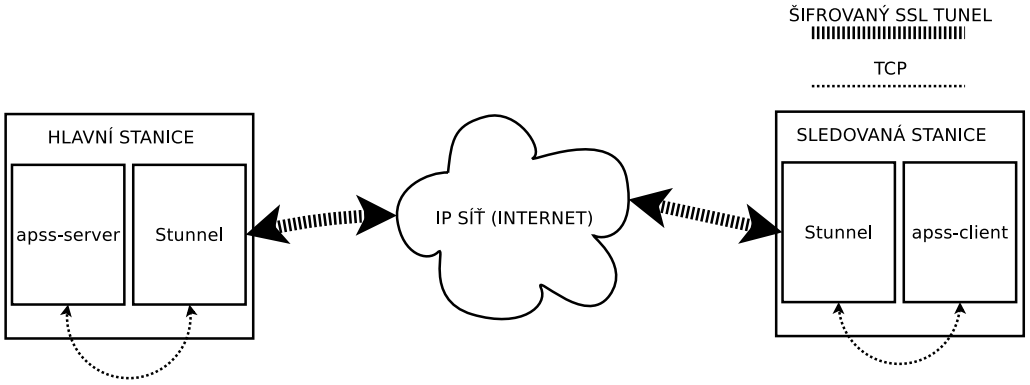
Pro klienta to znamená, že musí mít v konfiguračním souboru nastaveno:

```
1 ServerAdress = localhost
```

V konfiguračním souboru serveru je nutné povolit příjem dat od klienta localhost,

```
1 AllowClients = 127.0.0.1
```

jelikož data od všech klientů využívajících stunnel, jsou předávána přes toho rozhraní. Z tohoto důvodu je také nutná hlavička s IP adresou s jménem před získanými daty, neboť by jinak nešlo identifikovat, od kterého klienta data přišla. Filtrování klientů je fakt nutné nastavit například pomocí firewalu.



Obr. 4.1: Komunikace s využitím stunnelu.

4.6.2 Konfigurace HTTPS

K přenosu dat mezi webovým serverem a webovým prohlížečem uživatele se využívá nešifrovaný, textově orientovaný protokol HTTP. Data jsou přenášena jako prostý

text a je tedy možné je snadno odposlechnout a zneužít. K zašifrování dat přenášených mezi server a prohlížečem je možné použít protokol HTTPS, což je v podstatě HTTP obalený SSL vrstvou.

Aby bylo možné k webovému rozhraní přistupovat pomocí protokolu HTTPS, je nutné nejdříve správně nakonfigurovat webový server apache. Podobně jako u stunnelu bude potřeba vygenerovat certifikát. Nejprve je potřeba vytvořit adresář, ve kterém bude certifikát uložen.

```
1 # mkdir /etc/apache2/secret
2 # cd /etc/apache2/secret
```

Následně je možné vygenerovat soukromý klíč a žádost o vydání certifikátu, například pomocí příkazů:

```
1 # openssl genrsa -out /etc/apache2/secret/server.key 1024
2 # openssl req -new -key /etc/apache2/secret/server.key -out /etc/apache2/secret/
    server.csr
```

Poté je možné ze žádosti vygenerovat self-signed certifikát s platností jeden rok.

```
1 # openssl x509 -req -days 365 -in /etc/apache2/secret/server.csr -signkey /etc/
    apache2/secret/server.key -out /etc/apache2/secret/server.crt
```

Dalším krokem je zavedení modulu `mod_ssl`. To je možné udělat následujícím příkazem:

```
1 # a2enmod ssl
```

Dále je nutné zajisti aby server naslouchal na portu 443, jenž je vyhrazen pro HTTPS. To lze zajistit přidáním

```
1 Listen 443
```

do souboru `/etc/apache2/ports.conf`.

Poslední krok spočívá ve vytvoření nového virtuálního hosta. To lze zařídit například přidáním

```
1 NameVirtualHost *:443
2 <VirtualHost *:443>
3 DocumentRoot /var/www
4         SSLEngine On
5         SSLCertificateKeyFile /etc/apache2/secret/server.key
6         SSLCertificateFile /etc/apache2/secret/server.crt
7 </VirtualHost>
```

do souboru `/etc/apache2/sites-enabled/000-default`. Po restartu webového serveru a zadaní `https://<adresa_serveru>/` by měl být zajištěn přístup do webového rozhraní aplikace pomocí HTTPS. Tento postup byl převzat z [16].

5 POUŽÍVÁNÍ APLIKACE

Tato část práce představuje stručný uživatelský manuál. Popisuje jakým způsobem je možné spouštět klienta a server a hlavně pak možnosti a ovládání webového rozhraní.

5.1 Klient a server

Spouštět klienta či server je možné stejným způsobem jako každý jiný program zadáním jména programu do konzole a potvrzením příkazu, tedy například:

```
1 # apss-client
```

Další možností jak program spustit, je využíti startovacího skriptu pomocí něhož, jde program i ukončit případně restartovat. Použití je následující:

```
1 # /etc/init.d/apss-client start|stop|restart
```

Programy při spuštění nejsou schopny přebírat žádné parametry ze standardního vstupu (tzv. přepínače), veškeré konfigurační volby je nutné provést v příslušném konfiguračním souboru. Po spuštění se programy chovají jako běžné procesy.

5.2 Webové rozhraní

Webové rozhraní jsou ve své podstatě dynamické webové stránky obsahující formuláře, pomocí kterých je možné nastavit v jaké podobě budou naměřená data uživateli prezentována.

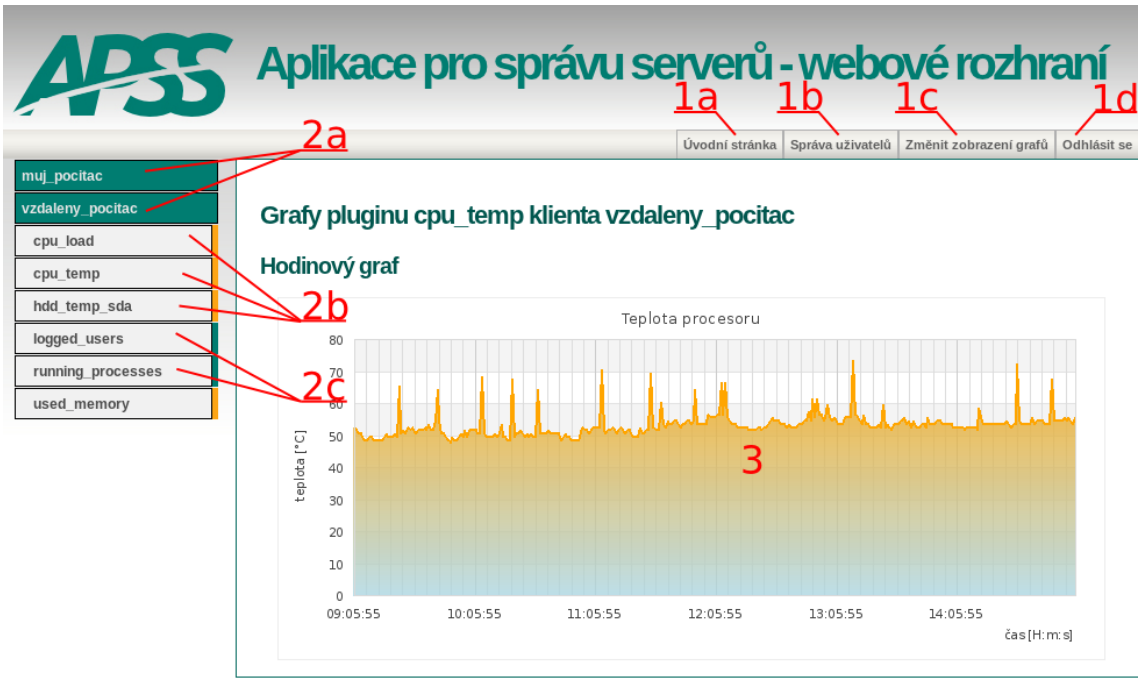
5.2.1 Přihlášení

Vstup do webového rozhraní se provádí skrze přihlašovací formulář, pomocí kterého se ověřuje identita vstupující osoby na základě znalosti přihlašovací jména a tajného hesla. Jinými slovy uživatel pro vstup do aplikace musí mít zřízen uživatelský účet. Po úspěšném ověření je mu pak umožněn vstup to aplikace.

5.2.2 Rozmístění ovládacích prvků

Webové rozhraní je rozčleněno do několika oblastí, přičemž každá má svůj specifický účel. Rozmístění ovládacích prvků ilustruje obrázek 5.1 .

1. Horizontální menu – tvoří lištu, která obsahuje
   (a) odkaz vedoucí na úvodní stránku,
   (b) odkaz vedoucí na formulář pro správu uživatelských účtů,



Obr. 5.1: Rozmístění ovládacích prvků ve webovém rozhraní.

(c) odkaz vedoucí na formulář pro nastavení grafů,
(d) odhlášení ze systému.

2. Vertikální menu – jedná se o rozbalovací menu obsahující seznam všech klientů a pluginů na nich spuštěných, kteří se serverem komunikují respektive mají záznam v databázi. Pro lepší rozlišení jsou numerické pluginy označeny oranžovým proužkem a textové zeleným.
   (a) jméno klienta,
   (b) číselné pluginy,
   (c) textové pluginy.

   Po kliknutí myši na položku reprezentující název klienta se menu rozbalí a zobrazí se seznam všech aktivních pluginů. Kliknutím na plugin se v závislosti na typu zobrazí buďto stránka s grafy nebo textový výpis.
3. Zobrazovací oblast – v této oblasti se zobrazují výsledky všech akcí inicializovaných uživatelem. V horní části se zobrazuje informační proužek, který informuje o výsledku provedené akce.

5.2.3 Správa uživatelů

Správa uživatelů je pouze základní a umožňuje uživateli s administrátorským účtem pomocí formulářů zobrazených na obrázku přidávat a mazat uživatelské účty běžných uživatelů, viz část 3.7.6. Při vytváření nového uživatele se do levého formuláře

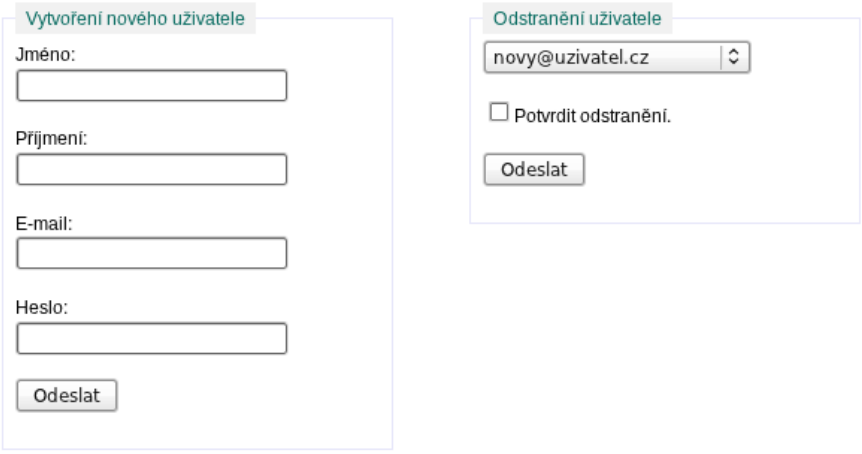
zadávají následující položky:

Jméno – křestní jméno nového uživatele.

Příjmení – příjmení jméno nového uživatele.

E-mailová – e-mailová adresa uživatele. Adresa musí být zadána ve správném formátu, jinak vytvoření uživatele není povoleno. E-mail slouží rovněž jako login.

Heslo – jedno se o dočasné heslo určené pouze pro první přihlášení. při vytváření uživatele zadá administrátor libovolné neprázdné heslo, systém nijak neřeší distribuci hesel novým uživatelům a je tedy plně v režii administrátora, jakým způsobem heslo novému uživateli předá.



Obr. 5.2: Formuláře pro zprávu uživatelů.

Po úspěšném odeslaní formuláře je vytvořen nový účet a je mu nastaven příznak „neaktivní“. Ten je nastaven do té doby, než se nový uživatel pokusí přihlásit. Při prvním přihlášení je vyzván k zadání nového hesla, tentokrát v přesně definované podobě (heslo musí být minimálně 6 znaků dlouhé a musí obsahovat minimálně jedno velké písmeno a jednu číslici). Proběhne-li změna hesla úspěšné, je již možné se pomocí nového hesla přihlásit.

Po vytvoření nového uživatele se jeho e-mailová adresa objeví v nabídce druhého formuláře, přes který je možné uživatele odstranit.

5.2.4 Zobrazení grafů

Přímo ve webovém rozhraní je možné měnit některé základní parametry grafů. Po kliknutí na položku *Změnit zobrazení* se zobrazí formulář pro výběr stanice - klienta, pod kterého měněný graf spadá. Po odeslání se zobrazí druhý formulář, jenž je zobrazen na obrázku[1] 5.3, zde je již možné provádět potřebné úpravy. Význam

[1]Formulář je na obrázku z úsporných důvodu rozdělen na dvě části.

položek v levé části je následující:

Vyberte plugin – výběr pluginu, pro jehož grafy se budou změny provádět.

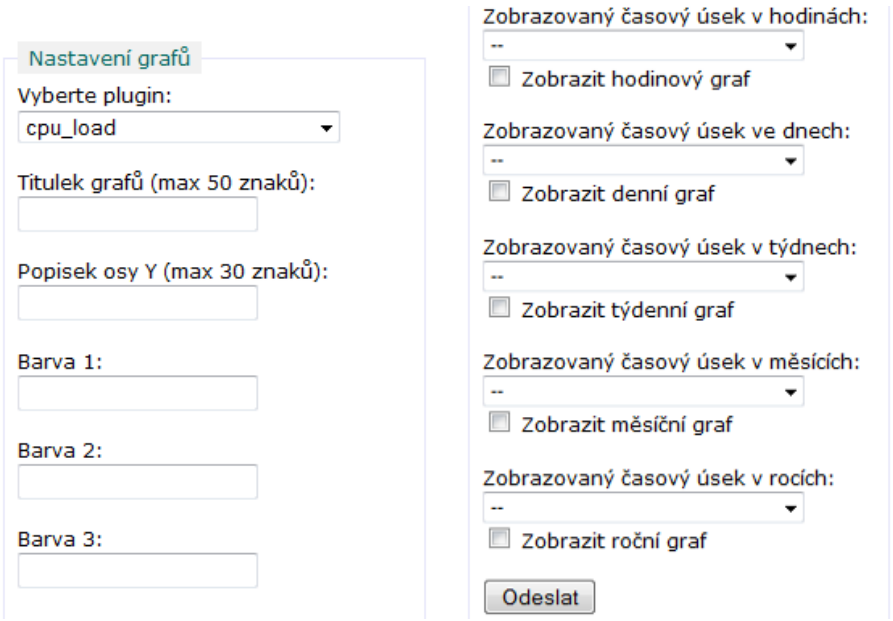
Titulek grafů – pole pro zadání nového titulku grafu, pokud není zadáno, použije se původní. Je možné zadat až 50 znaků. Změna se týká celé sady grafů.

Popisek osy Y – pole pro zadání popisku osy Y, pokud není zadáno, použije se původní. Je možné zadat až 30 znaků.

Barva 1 – definuje horní barvu přechodu, kterým je graf vyplněn. Zadat je možné název barvy (například red, blue apod., seznam definovaných barev je dostupný v dokumentaci knihovny jpgraph na [1]) nebo hexadecimální notaci barvy (například `#1E90FF`, `#fff` apod.). Průhlednosti je možné dosáhnout přidáním zavináče a stupně průhlednosti (od 0 do 1) za identifikátor barvy, tedy například `red@0.2`, `#1E90FF0.9` apod. Pokud není barva zadaná použije se původní.

Barva 2 – definuje spodní barvu přechodu, kterým je graf vyplněn. Formát zadávaní je stejný jako u Barvy 1.

Barva 3 – definuje barvu čáry. Formát zadávaní je stejný jako u Barvy 1.



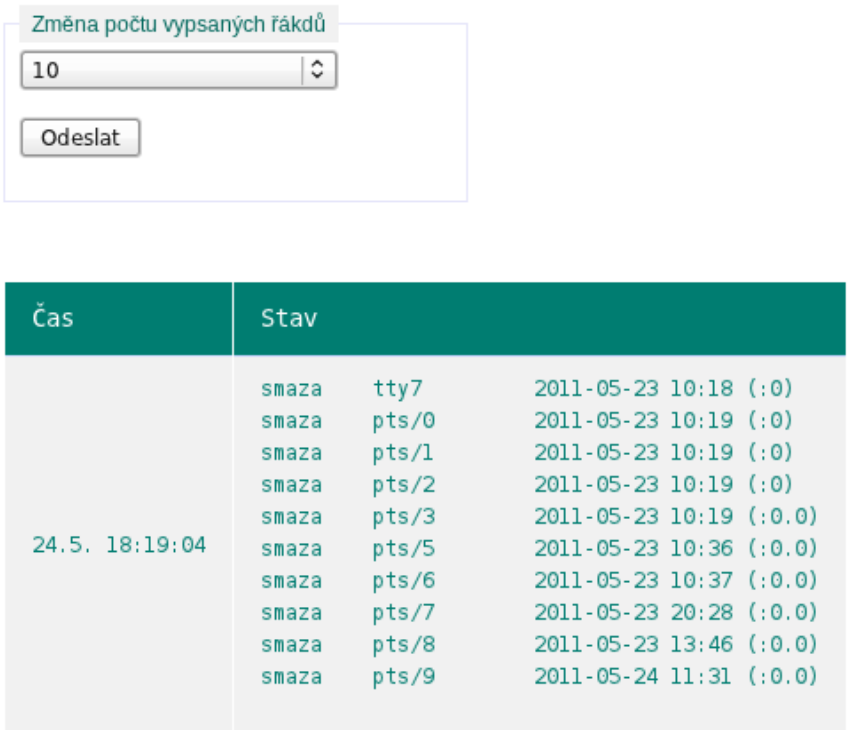
Obr. 5.3: Formulář pro nastavení grafů .

Další položky (pravá část obrázku) už se přímo netýkají grafické podoby grafů, ale určují, které grafy se budou zobrazovat a jak velký časový interval bude v příslušném grafu zobrazen. Není-li časový interval zadán, použije se výchozí nebo předchozí zadaná hodnota. Při výběru zobrazovaných grafů musí být zvolen minimálně jeden graf, pokud není vybrán žádný graf, formulář nelze odeslat.

V databázi je uložen výchozí záznam, který definuje všechny tyto parametry, a tím i výchozí vzhled grafů daného pluginu. Dojde-li k editaci byť jen jediného parametru, je vytvořen pro příslušný plugin nový záznam a ten je nadále používán.

5.2.5 Zobrazení textových výpisů

U textových výpisů je možné změnit počet vypisovaných záznamů. Tato volba je přístupná přímo u výpisu příslušného textové pluginu. Zvolená hodnota se ukládá do relace (session) a je platná pouze po dobu přihlášení. Výchozí hodnota je 10 výpisů. Podoba textového výpisu včetně formuláře pro změnu počtu vypisovaných záznamů je zobrazena na obrázku 5.4



Obr. 5.4: Jedna položka textového výpisu (plugin – `logged_users`).

6 TESTOVÁNÍ APLIKACE

Aplikace byla během vývoje testována pouze lokálně, což se nakonec ukázalo jako chyba. Při závěrečném testování v počítačové síti se objevilo několik nedostatků, které se při lokálním testování neprojevily. Nakonec byly všechny problémy vyřešeny a test proběhl úspěšně.

6.1 Parametry závěrečného testu

Aplikace byla testována na dvou stanicích nacházejících se v LAN síti. Na jedné stanici byl spuštěn server a klient, na druhé pak pouze klient s celkem jedenácti spuštěnými pluginy (viz tabulka 6.1). Z časových důvodů byl interval odesílání zvolen 5 sekund, data tedy byla získávána 12× rychleji, než by tomu bylo při výchozím jednominutovém intervalu. Rovněž intervaly přepočítávání byly 12× zkráceny. Test trval přibližně 20 hodin. Aby bylo shromážděno co nejvíce dat nebylo aktivováno promazávaní záznamů.

vzdaleny_pocitac		
jméno pluginu	typ pluginu	sledovaná veličina
cpu_load	numerický	zátěž procesoru v %
cpu_temp	numerický	teplota procesoru
hdd_temp_sda	numerický	teplota pevného disku
used_memory	numericky	zaplnění operační paměti
logged_users	textový	přihlášení uživatelé
netstat_inet	textový	výpis spuštěných služeb
running_processess	textový	výpis spuštěných procesů
muj_pocitac		
cpu_load	numerický	zátěž procesoru v %
used_memory	numericky	zaplnění operační paměti
logged_users	textový	přihlášení uživatelé
netstat_inet	textový	výpis spuštěných služeb

Tab. 6.1: Pluginy spuštěné na testovacích stanicích.

6.2 Výsledky testu

Z tabulky 6.2 je patrné, že z celkového objemu 167 MiB dat v databázi tvoří největší podíl (přibližně 97%) hodnoty textových pluginů. Zbylé 3% objemu dat připadají

na hodnoty numerických pluginů včetně přepočtených hodnot. Z toho vyplývá, že velikost databáze silně ovlivní vysoký počet textových modulů, přestože budou každý den jejich hodnoty odmazávány.

Řešením by bylo zvětšit interval odesílání například na 5 minut, což by se však projevilo i na numerických pluginech. Další možností je vytvořit dva programy klienta pokaždé s jiným názvem a cestou ke konfiguračnímu souboru a každý používat na jeden typ pluginů s odlišným intervalem odesílání.

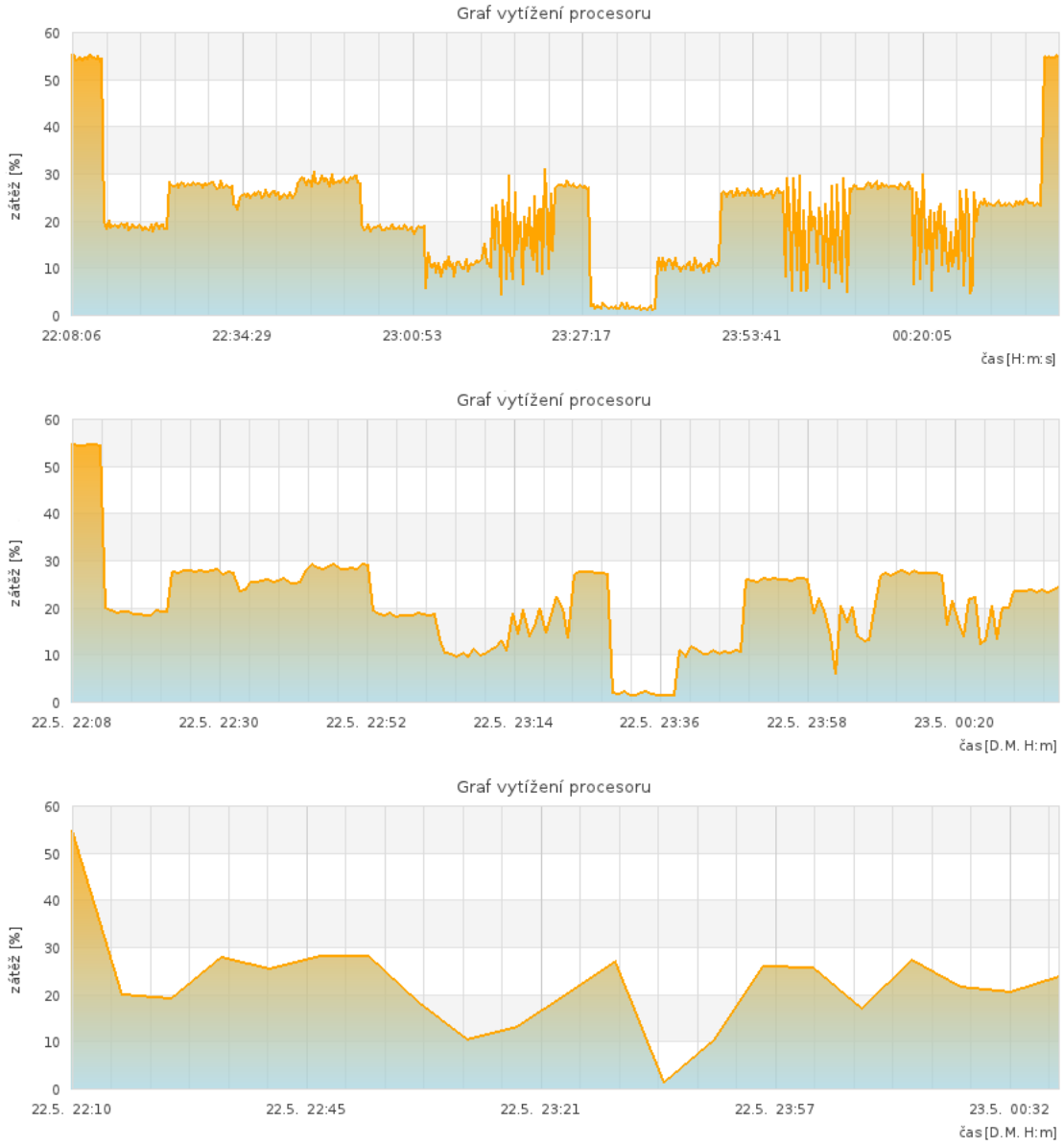
Po ukončení testu byla pomocí zálohovacího skriptu provedena záloha testovací databáze. Velikost archivovaného souboru, jenž obsahoval kompletní zálohu všech dat, byla 1,2 MB, což představuje 0,7% velikosti databáze. Provádění záloh každý den tady výrazně neovlivní kapacitu pevného disku.

Tabulka	Záznamů	Velikost
clients	11	48,0 KiB
graphs	4	16,0 KiB
string_values	39 235	162,1 MiB
number_values	46 300	4,0 MiB
day_number_values	9 508	672,0 KiB
week_number_values	1 228	80,0 KiB
month_number_values	132	32,0 KiB
year_number_values	12	32,0 KiB
9 tabulek	96 431	167,0 MiB

Tab. 6.2: Statistika databáze po 20 hodinovém testu.

6.2.1 Grafy

Použití mediánu pro přepočet hodnot se ukázalo jako velmi vhodná volba, což dokazují grafy na obrázku 6.1. Pro názornost všechny grafy zobrazují stejný časový interval. Jako první zobrazen poměrně detailní hodinový graf, dále pak graf denní, na kterém je jasně patrna zachovaná převažující tendence a odfiltrování extrémních hodnot. Objem dat pro zobrazení stejného časové intervalu je však 5× menší než u hodinového grafu. Poslední, týdenní graf, je generován z objemu dat přibližně 40× menšího než hodinový graf. Přestože je rozlišení grafu již velmi nízké stále, je převažující tendence patrná. Podobně platí tyto vztahy i pro ostatní grafy.



Obr. 6.1: Trojice grafů (hodinový, denní, a týdenní) vygenerovaná při testování.

7 ZÁVĚR

Cílem této práce bylo podrobněji se seznámit s problematikou počítačových serverů, získat informace a poznatky o jejich fungování a správě. Dále pak získat přehled o technikách a nástrojích, pomocí kterých je možné správu počítačových serverů zajistit. Důraz byl kladen především na možnosti monitorování linuxových serverů a na základě nabitých vědomostí navrhnout a realizovat vlastní aplikaci pro monitorování serverů.

Jelikož možností jak aplikaci realizovat existuje celá řada, byly vybrány technologie, jejichž schopnosti a možnosti jsou prověřeny mnoha tisící uživateli po celém světě. Přihlíženo bylo i k osobním zkušenostem a důležitou roli hrála také dostupnost a kvalita informačních zdrojů, ze kterých bylo možné informace o konkrétním produktu čerpat. Dále bylo nutné, aby vybrané technologie byly kompatibilní s platformou Gnu/Linux a aby byly šířeny jako open source.

Při návrhu aplikace byly zohledněny požadavky na snadnou konfiguraci, snadné rozšíření, rychlost a nízké systémové nároky. Aplikace byla navržena a implementována jako celek skládající se z několika vzájemně propojených částí.

Základ systému tvoří aplikace typu klient-server. Klient je tvořen jádrem a zásuvnými moduly, které získávají požadovaná data. Každý uživatel si může snadno přidat vlastní modul, čímž je zajištěna snadná rozšiřitelnost aplikace. Oba programy jsou naprogramovány v jazyce C++ a spotřebovávají velmi malé množství systémových prostředků.

Aplikace je zabezpečena na několika vrstvách. Zamezuje tak útočníkovi proniknout do jakékoliv její části. K zabezpečení byly použity ověřené postupy a mechanismy jako například vhodně omezená práva spuštěných programů nebo použití protokolu SSL při přenosu dat sítí či přístupu k webovému rozhraní.

Pro tvorbu grafů je využita knihovna JPGaph, jejíž možnosti jsou velmi široké a generované grafy mají vysokou grafickou úroveň.

Při závěrečném testování aplikace se objevilo několik problémů, které zůstaly při vývoji skryty. Všechny se ale podařilo uspokojivě vyřešit a aplikace je tedy plně funkční. Testovaní však probíhalo jen relativně krátkou dobu a s malým počtem stanic. Proto nemusí některé předpoklady při dlouhodobém chodu aplikace platit. Pro důkladné otestování by aplikace musela běžet několik měsíců v reálném prostředí, což nebylo v tomto případě především z časových důvodů možné.

LITERATURA

[1] Asial.Co.Ltd. *JpGraph - Most powerful PHP-driven charts* [online]. 2010, [cit. 4. 12. 2010]. Dostupné z URL: <http://jpgraph.net/>.

[2] BARLETTA, M. *Správa serverů v grafickém módu* [online].[cit. 3. 12. 2010]. Dostupné z URL: <http://www.novellenterprise.cz/sprava-serveru-v-grafickem-modu>.

[3] ČÍHAŘ, M. *Munin – monitorování serverů* [online]. 25.4.2006 06:00 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.linuxsoft.cz/article.php?id_article=1203>.

[4] DOČEKAL, M. *Správa linuxového serveru: S.M.A.R.T. a zdraví pevných disků* [online]. Čtvrtek, 4. únor 2010 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-smart-a-zdravi-pevnych-disku>

[5] DOSTÁL, R. *Sokety a C++* [online]. 04.11. 2002 [cit. 13. 12. 2010]. Dostupné z URL: <http://www.builder.cz/art/cpp/sokety_a_cpp.html>.

[6] FIELDING, R. *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1* [online]. IETF, Network Working Group, June 1999 [cit. 2. 12. 2010]. Dostupné z URL: <http://tools.ietf.org/html/rfc2616>.

[7] GIGEL, M. *Seriál MRTG - grafické prehľady* [online]. 18. 12. 2001 10:01 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.root.cz/serialy/mrtg-graficke-prehlady/>.

[8] GODARD, S. *SYSSTAT* [online]. [cit. 3. 12. 2010]. Dostupné z URL: <http://sebastien.godard.pagesperso-orange.fr>.

[9] HÄRING, D. *Monitorování zátěže systému* [online]. 17. 06. 2002, 12:00 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.linuxzone.cz/index.phtml?ids=9&idc=279>.

[10] honzikweb1.estranky.cz *Linux: monitorování systému, procesy, příkaz ps, adresář /proc, obsah adresáře /var* [online]. 12. 6. 2007 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.honzikweb1.estranky.cz/clanky/matrose/SOS17.html>.

[11] ISOTTON, A. *C++ dlopen mini HOWTO* 2006-03-16 [cit. 14. 5. 2011]. Dostupné z URL: <http://www.isotton.com/devel/docs/C++-dlopen-mini-HOWTO/C++-dlopen-mini-HOWTO.html>.

[12] Lm-sensors *Lm_sensors - Linux hardware monitoring* [online]. [cit. 3. 12. 2010]. Dostupné z URL: <http://www.lm-sensors.org>.

[13] MARTÍNEK, D. *Zásuvné moduly - pluginy* [online]. Poslední modifikace: 24. října 2010. [cit. 12. 12. 2010]. Dostupné z URL: <http://www.fit.vutbr.cz/ martinek/clang/plugins.html>.

[14] Nagios Enterprises *Welcome To Nagios – The Industry Standard In Open Source Monitoring* [online]. 2010 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.nagios.org/>.

[15] NĚMEC, J. *C/C++ (31) - Jazyk C++, historie, charakteristika, vztah k C* [online]. 9.1.2006 06:00, [cit. 4. 12. 2010]. Dostupné z URL: <http://www.linuxsoft.cz/article.php?id_article=1058>.

[16] PELKA, T. *MNSB cvičení 7.* 20.3.2011 [cit. 22. 5. 2011]. Dostupné z URL: <http://srv-126-boo.utko.feec.vutbr.cz/pa126/2011/03/20/mnsb-cviceni-7/>.

[17] POMIKÁLEK, J. *Sledování sítě (SNMP, MRTG, Nagios)* [online]. 26-Nov-2003 10:01 [cit. 3. 12. 2010]. Dostupné z URL: <http://www.fi.muni.cz/ kas/p090/referaty/2003-podzim/skupina10/snmp.html>.

[18] SOBORKA, J. *Profesionální grafy v PHP snadno a rychle* [online]. 29. 05. 2002, [cit. 4. 12. 2010]. Dostupné z URL: <http://interval.cz/clanky/profesionalni-grafy-v-php-snadno-a-rychle/>.

[19] STONES, R., MATTHEW, N. *Linux začínáme programovat* Odpovědný redaktor Ivo Magera; překlad Jan Škvařil. 2. vyd. Praha : Computer Press, 2000. 897 s. ISBN 80-7226-307-2, Prodejní kód: K0338.

[20] Sun Microsystems, Inc. *Chapter 13. Storage Engines* [online]. 2008, Posted by Adrian Singer on January 26 2008 1:15pm [cit. 4. 12. 2010]. Dostupné z URL: <http://dev.mysql.com/doc/refman/5.0/en/storage-engines.html>.

[21] The PHP Group. *Historie PHP* [online]. Last updated: Sat, 24 Mar 2007 [cit. 4. 12. 2010]. Dostupné z URL: <http://php.tonnikala.org/manual/cs/history.php#history.php>.

[22] TOMICKI, L. *Clog* Last update: Saturday, 04th December, 2010 [cit. 14. 5. 2011]. Dostupné z URL: <http://tomicki.net/clog.php>.

[23] trac.steve.museum *MedianUDF* [cit. 17. 5. 2011]. Dostupné z URL: <http://trac.steve.museum/wiki/MedianUDF>.

[24] VIKLUND, A. *Free website templates* 2011 [cit. 17. 5. 2011]. Dostupné z URL: <http://andreasviklund.com/templates/>.

[25] WAGNER, R. *Configuration File Reader for C++* 26 May 04 [cit. 14. 5. 2011]. Dostupné z URL: <http://www-personal.umich.edu/ wagnerr/ConfigFile.html>.

[26] WIEËRS, D. *Dstat: Versatile resource statistics tool* [online]. Last modified on: Sun 24 October 2010 [cit. 3. 12. 2010]. Dostupné z URL: <http://dag.wieers.com/home-made/dstat/>.

[27] weboveaplikace.info *Uložené procedury (stored procedures) v MySQL* 25. Leden 2008 [cit. 22. 5. 2011]. Dostupné z URL: <http://weboveaplikace.info/2008/01/25/procedury-v-mysql/>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

HTTP Hypertext Transfer Protocol

SMTP Simple Mail Transfer Protocol

TCP Transmission Control Protocol

SSH Secure Shell

DMA Direct Memory Access

S.M.A.R.T. Monitoring Analysis and Reporting Technology

SMNP Simple Network Management Protokol

NMS Network Management System

GSM Global System for Mobile Communications, původně Groupe Spécial Mobile

RRD Round Robin Database

MRTG Network Management System

PHP PHP: Hypertext Preprocessor

UDF User defined functions

HTML HyperText Markup Language

CSS Cascading Style Sheets

JS Javascript

SEZNAM PŘÍLOH

A Obsah přiloženého DVD **74**

B Souborový systém `/proc` **75**

C Souborový systém `/var` **76**

D Třída `client` **77**

E Třída `server` **79**

A OBSAH PŘILOŽENÉHO DVD

Obsah DVD:

- Obraz virtuálního disku pro virtualizační nástroj VirtualBox s kompletně funkční instalací aplikace.
- Veškeré zdrojové kódy potřebné k instalaci aplikace aplikace.
- Elektonická verze dokumentu – pdf s touto prací a zdrojové *.tex* soubory.
- Seznam všech nainstalovaných balíčku operačního systému Debian GNU/Linux

B SOUBOROVÝ SYSTÉM /PROC

/proc/1	Adresář s informací o procesu číslo 1. Každý z procesů má v adresáři /proc vlastní podadresář, kterého jméno je stejné, jako identifikační číslo procesu.
/proc/cpuinfo	Různé informace o procesoru. Například typ, výrobce model, atd.
/proc/devices	Seznam ovladačů zařízení konfigurovaných pro aktuálně běžící jádro systému.
/proc/dma	Informuje o tom, které kanály DMA jsou právě využívány.
/proc/filesystems	Souborové systémy konfigurované v jádru systému.
/proc/interrupts	Informuje o tom, která přerušení jsou využívána, i o historii žádostí o využití každého z nich.
/proc/ioports	Informuje o tom, který z vstupně-výstupních portů se momentálně využívá.
/proc/kcore	Obraz fyzické paměti systému. Má velikost odpovídající velikosti fyzické paměti systému. Ve skutečnosti ale samozřejmě nezabírá takovéto množství paměti, protože jde o soubor generovaný „na požádání“, tedy pokaždé jenom v okamžiku, kdy k němu různé programy přistupují. Uvědomte si, že soubory souborového systému /proc nezabírají ve skutečnosti (než je zkopírujete na nějaké jiné místo na disku) vůbec žádný diskový prostor.
/proc/kmsg	Výstupní hlášení jádra systému. Zde uložená hlášení využívá i program syslog.
/proc/ksyms	Tabulka symbolů jádra systému.
/proc/loadavg	Statistika zatížení systému – tři celkem nic neříkající indikátory toho, kolik práce systém momentálně má.
/proc/meminfo	Informace o využití paměti, jak fyzické, tak virtuální (swap).
/proc/modules	Informuje o tom, které moduly jádra jsou právě zavedeny v paměti.
/proc/net	Informace o stavu síťových protokolů.
/proc/self	Symbolický link do adresáře procesů toho programu, který zrovna přistupuje k souborovému systému /proc. Když k systému souborů /proc současně přistupují dva různé procesy, budou mít přidělené dva různé linky. Tímto způsobem se mohou programy pohodlně a jednoduše dostat k vlastnímu adresáři.
/proc/stat	Různé statistiky týkající se systému. Např. počet chyb stránkování během zavádění systému a podobné.
/proc/uptime	Informuje o tom, jak dlouho systém běží.
/proc/version	Verze jádra systému.

Tab. B.1: Přehled některých významných adresářů souborového systému /proc.

C SOUBOROVÝ SYSTÉM /VAR

/var/catman	Vyrovnávací paměť pro manuálové stránky, které jsou formátované na požádání. Zdroje pro tyto manuálové stránky jsou obvykle uložené v adresáři /usr/man/man*. Manuálové stránky v předem formátované verzi jsou uložené v adresáři /usr/man/cat*. Manuálové stránky je běžně třeba při prvním prohlížení formátovat. Formátované verze jsou pak uložené právě v adresáři /var/catman. Další uživatel, který si chce stejné stránky prohlížet, tak nemusí čekat na jejich opakované formátování. (Soubory v uvedeném adresáři /var/catman je obvykle potřeba mazat, stejně jako dočasné soubory v adresářích /tmp a /var/tmp.) /var/lib Soubory, které se při normálním provozu systému mění. /var/local Měnící se data pro programy instalované v adresáři /usr/local (tj. programy instalované správcem systému). Upozorňujeme, že lokálně instalované programy by měly používat i ostatní podadresáře nadřazeného adresáře /var, např. /var/lock.
/var/lock	Soubory tzv. zámků. Většina programů dodržuje určitou konvenci a vytváří v adresáři /var/lock zámky. Tím dávají ostatním programům najevo, že dočasně využívají některé zařízení nebo soubor. Jiné programy, které by chtěly stejné zařízení či soubor ve stejném okamžiku používat, se o to nebudou pokoušet.
/var/log	Adresář obsahuje tzv. log-soubory různých programů. Například program login zaznamenává (do souboru /var/log/wtmp) všechna přihlášení a odhlášení uživatelů systému, program syslog ukládá (do souboru /var/log/messages) všechny hlášky jádra systému a systémových programů. Velikost souborů v adresáři /var/log dost často nekontrolovaně roste. Proto se musí v pravidelných intervalech mazat.
/var/run	Adresář, do něhož se ukládají soubory obsahující informace o systému, jež platí až do jeho dalšího zavedení. Tak například soubor /var/run/utmp obsahuje informace o současně přihlášených uživatelích systému. /var/spool Adresáře pro elektronickou poštu, systém „news“, tiskové fronty a další subsystémy, které využívají metodu „spool“ a princip řazení úloh do fronty. Každý z těchto subsystémů má v tomto adresáři svůj vlastní podadresář, např. poštovní schránky uživatelů jsou uložené v podadresáři /var/spool/mail.
/var/tmp	Do adresáře /var/tmp se ukládají velké dočasné soubory a dočasné soubory, které budou existovat déle než ty, které se ukládají do adresáře /tmp. (Avšak správce systému by měl dbát na to, aby stejně jako v adresáři /tmp, ani v adresáři /var/tmp nebyly uloženy velmi staré dočasné soubory.)

Tab. C.1: Přehled některých významných adresářů souborového systému /var.

D TŘÍDA CLIENT

```
1  /*
2   *       Soubor:  client.h
3   *       Datum:   12.05.2011
4   *       Autor:   Petr Smahel, xsmahe00@stud.feec.vutbr.cz
5   *       Projekt: Diplomová práce na téma: Aplikace pro správu serverů
6   *       Popis:   Hlavičkový soubor třídy client, která zajišťuje hlavní funkčnost
        klietnského programu.
7   */
8
9  #ifndef CLIENT_H
10 #define CLIENT_H
11
12 #include <iostream>
13 #include <string>
14 #include <vector>
15 #include <iostream>
16 #include <sstream>
17 #include <unistd.h>
18 #include <netdb.h>
19 #include <netinet/in.h>
20 #include <dlfcn.h>
21 #include <sys/types.h>
22 #include <sys/socket.h>
23 #include <cstring>
24 #include "log.h"
25 #include "except.h"
26 #include "plugin.h"
27 #define BUFSIZE 100
28
29 using std::string;
30 using std::cout;
31 using std::endl;
32 using std::cerr;
33 using std::istringstream;
34 using std::ostringstream;
35
36 //Struktura pro nacteni pluginu
37 typedef struct
38 {
39         void* handle;
40         create_t* creator;
41         destroy_t* destroyer;
42         plugin* plug;
43 }MODUL;
44
45 class Client
46 {
47         private:
48                 string out, in;          // Odesilany a prijimany text
49                 hostent *host;           // Vzdaleny pocitac
50                 sockaddr_in serverSock; // Vzdaleny "konec potrubi"
51                 int mySocket;            // Soket
52                 int port;                // Cislo portu
53                 char adresa[30];         // adresa
54                 char buf[BUFSIZE];       // Prijimaci buffer
55                 int size;                // Pocet prijatych a odeslanych bytu
```

```
56                  bool logging;           // Logovani ane/ne
57                  string loggingfile;     // Cesta k logovacimu souboru
58
59          public:
60                  Client(const string adr, int prt, bool log, std::string logfile);
61                  void Connect();
62                  void DisConnect();
63                  void Send(string data);
64                  void Recv();
65                  string GetRecv();
66                  void LoadPlugin(const std::string &cesta, const std::string &
                        loadplug, std::vector<MODUL> &moduly);
67                  string ConvertIntToString(int number);
68                  ~Client();
69
70  };
71  #endif                                                          /* CLIENT_H
        */
```

E TŘÍDA SERVER

```
1  /*
2   *       Soubor:  server.h
3   *       Datum:   12.05.2011
4   *       Autor:   Petr Smahel, xsmahe00@stud.feec.vutbr.cz
5   *       Projekt: Diplomová práce na téma: Aplikace pro správu serverů
6   *       Popis:    Hlavičkový soubor třídy server, která zajišťuje hlavní funkčnost
        serverového programu.
7   */
8
9
10 #ifndef SERVER_H
11 #define SERVER_H
12 #include <errno.h>
13 #include <iostream>
14 #include <iterator>
15 #include <fstream>
16 #include <sstream>
17 #include <signal.h>
18 #include <string>
19 #include <algorithm>
20 #include <vector>
21 #include <unistd.h>
22 #include <netdb.h>
23 #include <netinet/in.h>
24 #include <sys/types.h>
25 #include <sys/socket.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <cstring>
29 #include <arpa/inet.h>
30 #include "except.h"
31 #include "log.h"
32 //velikost přijimacího bufferu
33 #define BUFSIZE 10000
34
35
36
37 using std::string;
38 using std::cout;
39 using std::endl;
40 using std::cerr;
41 using std::ostringstream;
42 using std::istringstream;
43 using std::vector;
44 class Server
45 {
46         private:
47                 string text;            // Prijimany text
48                 sockaddr_in sockName;   // "Jmeno" portu
49                 sockaddr_in clientInfo; // Klient, ktery se pripojil
50                 int mainSocket;         // Soket
51                 int port;               // Cislo portu
52                 char buf[BUFSIZE];      // Prijimaci buffer
53                 int size;               // Pocet prijatych a odeslanych bytu
54                 socklen_t addrlen;      // Velikost adresy vzdaleneho pocitace
55                 int klient;             // Identifikátor socketu klienta
```

```
56                  bool logging;           // Logovani ane/ne
57                  string loggingfile;     // Cesta k logovacimu souboru
58
59
60          public:
61                  Server(int p,bool log, std::string logfile);
62                  void setup();
63                  void accpt();
64                  void rcv();
65                  void snd();
66                  void cls();
67                  bool checkAdrr();
68                  ~Server();
69                  string getRecv();
70                  string getClientAdress();
71                  string convertIntToString(int number);
72                  bool checkAddr(std::vector<string>& allowClients);
73                  void splitString(const string& s, char c, vector<string>& v);
74                  string getHostNameFromAdrr(string adrr);
75                  };
76
77
78
79
80  #endif /* SERVER_H */
```