Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Flexibilní logování pro embedded Linuxové systémy

David Vavřička

Vedoucí práce: Ing. Matěj Laitl

12. dubna 2016

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. dubna 2016 .....

Odkaz na tuto práci

Vavřička, David. *Flexibilní logování pro embedded Linuxové systémy.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Doplňte

Klíčová slova logování, vestavěné systémy, logovací démoni, Linux, Rsyslog

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords logging, embedded systems, logging daemons, Linux, Rsyslog

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Kapitola 1

Technické požadavky

Cílem je upravit logovací řešení pro set-top box EKT DID7006mTF [1] tak, aby splňovalo technické požadavky popsané v této kapitole. Řešení musí fungovat a být otestováno na zmíněném modelu set-top boxu a pokud možno by mělo být přenositelné i na jiné typy set-top boxů. Požadavky jsou rozděleny na základní a rozšířené. Rozšířené požadavky není nutno implementovat.

1.1 Základní technické požadavky

1.1.1 Snížení objemu logů

Je žádoucí umožnit snížit objem zasílaných logů z důvodu přílišného zatížení sítě a serverových disků. A to tak, že zadavatel bude schopen nadefinovat pro každou komponentu úroveň severity zpráv, od které mají být posílány na server. Výchozí nastavení dodá zadavetel.

1.1.2 Vzdálená konfigurace

Technické řešení musí být schopno za běhu pomocí SHELL-ového API měnit minimální severitu zpráv pro jednotlivé komponenty a dále toto API musí umožnit nastavit výchozí severitu, která se použije pro komponenty ji nemají explicitně nastavenou. Takto změněné nastavení musí být perzistentní i po restartu STB. Výchozí nastavení se obnoví až po factory resetu. API navrhne dle své libovůle sám řešitel.

1.1.3 Rate-limiting odesílaných zpráv

Nové logovací řešení musí být schopné provádět rate-limiting odesílanych zpráv tak, aby nepřekročilo maximální vyhrazenou šířku pásma. Bude umožněno nastavit jak dlouhodobé tak krátkodobé limity. Naivní rate-limiting je i v existujícím řešení, řešitel navrhne výchozí nastavení nového řešení tak, aby přibližně odpovídalo současnému chování.

1.1.4 Formát logů

Je nutno zachovat formát logů jako ho má původní řešení, aby se jednalo o drop-in replacement bez nutnosti jakkoli měnit konfiguraci serveru, který sbírá logy od set-top boxů.

1.1.5 Razítkování zpráv

Každé zprávě se musí přidat textový prefix id=N, kde N monotonicky roste s každou zprávou. To slouží pro detekci ztracených zpráv. Id přeteče po 32 nebo 64 bitech, to záleží na rozhodnutí řešitele. Po rebootu STB id znovu začíná od 1.

1.1.6 Post-processing zpráv

Zadavatel má pouze částečnou kontrolu nad zprávami generovanými aplikacemi na set-top boxu, například nedokáže ve všech případech eliminovat dlouhé prefixy u zpráv. Je proto nutno takové prefixy rozpoznat a vhodně odfiltrovat před odesláním. Ze stejného důvodu mají některé zprávy nevhodně vyplněnou severitu a položku app-name. Jejich správné hodnoty jsou uloženy v textu zprávy, jejíž formát je pro jednotlivé skupiny zpráv konstantní. Řešení bude schopné tyto údaje z těla zprávy extrahovat a nahradit jimi původní metadata. Tato pravidlo musí být možné definovat a měnit bez nutnosti nového sestavení softwaru. Řešitel vytvoří pro ukázku 3 pravidla, která budou sloužit zadavateli jako šablony pro možná budoucí filtrovací pravidla. Tato pravidla jsou popsána v kapitole analýza.

1.2 Rozšířené technické požadavky

1.2.1 Komprese zpráv

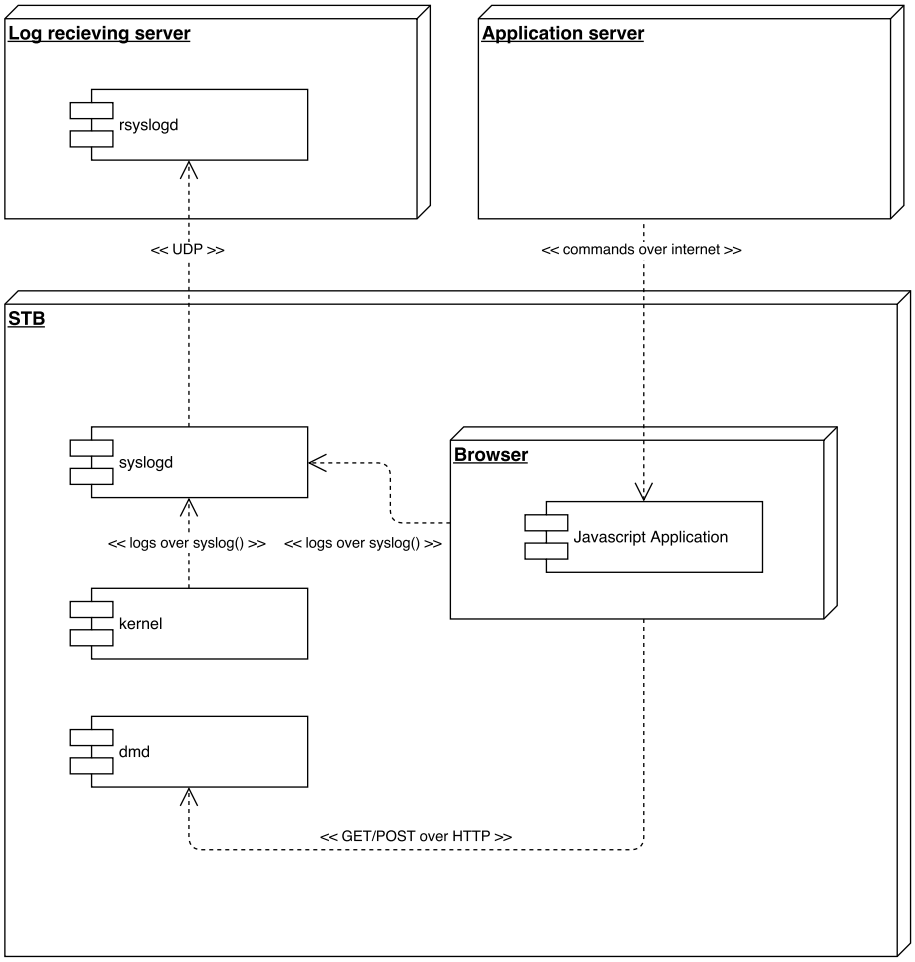
Bylo by vhodné zvážit pro a proti komprese zpráv. Vyplatí se ušetřená přenesená data oproti režiji spojené s kompresí a dekompresí zpráv?

1.2.2 Integrace do existující servisní C++ komponenty

Zadavatel na STB provozuje malého démona dmd napsaného v C++, který mimo jiné obsahuje minimalistický HTTP webserver. Dále v browseru běží Javascript aplikace (nangu.TV portál), která pomocí messagingu komunikuje s centrálním serverem. Tato Javascript aplikace ovšem nemůže přímo používat Shell API. Požadavkem je rozšířit C++ komponentu dmd tak, aby umožnila Javacript aplikaci řídit konfiguraci logování (viz bod Vzdálená konfigurace).

Kapitola 2

Analýza a návrh



Obrázek 2.1: Diagram nasazení původního řešení STB

2.1 Současné řešení

V této kapitole jsou uvedeny a popsány jednotlivé pro tuto práci důlužité komponenty současného řešení.

2.1.1 Runtime prostředí

Na STB běží OS GNU/Linux s Busybox sadou aplikací, která nahrazuje standardní GNU shellové utility. Busybox aplikace jsou díky své malé velikosti vhodnější pro embedded zařízení. Na druhou stranu neobsahují všechny funkcionality a možnosti nastavení.

2.1.1.1 Browser

Browser je jednoduchý internetový prohlížeč dodaný výrobcem STB. V něm běží v Javascriptu zadavatelem napsaná aplikace poskytující uživatelské rozhraní. Mimo to také naslouchá zprávám z aplikačního serveru, na jejichž základě může provádět akce, a to včetně generování POST či GET requestu pro démon dmd. Tato javascriptová aplikace spolu s dalším podpogramem nesoucím název player generují velké množství logů. Tyto logy nepoužívají standardizovanou syslog množinu severit, ale svou vlastní s ní nekompatibilní. Logovací démon ovšem očekává zprávy právě podle syslog standardu. Implementaci browseru ani playeru nemůže zadavatel měnit a proto je třeba nekompatibilitu vyřešit pomocí logovacího démonu.

2.1.1.2 Servisní komponenta dmd

Dmd je v C++ napsaná servisní komponenta, která zprostředkovává komunikaci mezi Browserem a shellovým prostředím pomocí minimalistického HTTP serveru. Umožňuje JavaScript aplikaci na STB provádět operace, které jsou dostupné jen z shellového API.

2.1.1.3 BusyBox Syslogd

Jedná se o minimalistický logovací démon, který je podrobněji popsán v následující kapitole „Srovnání logovacích démonů“. Zadavatel mu pro své potřeby navíc doimplementoval číslování jednotlivých zpráv pro rozpoznání výpadku a za druhé naivní rate-limiting zpráv. Ten je naivní z důvodu, že používá statický buffer na počet zpráv, který nebere ohled na jejich velikost a hlavně bufferované zprávy se neodešlou samy, ale jen s nově příchozími zprávami. Za použití tohoto démonu není možné splnit některé stěžejní technické požadavky, jako například „Snížení objemu logů“ nebo „Post processing zpráv“. Proto bude nutné ho nahradit nějakým vyspělejším démonem. Tím se ale zabývá až další kapitola „Nové řešení“.

2.1.2 Aplikační server

Aplikační server se stará o business logiku, například poskytuje uživateli práva přehrát daný kanál nebo film, komunikuje s javascriptovou aplikací, která se vůči němu autorizuje a podobně. AS není možné v této práci jakkoli měnit. Je nutné zachovat kompatibilitu.

2.1.3 Servery pro sběr logů

Zadavatel provozuje cluster serverů sbírajících logy od statisíců STB. Běží na nich rsyslog nakonfigurovaný pro co nejlepší výkonost.

2.2 Vývoj pro STB

2.2.1 Buildovací systém Gu

Gu je buildovací systém pro linuxová embedded zařízení, který si klade za cíl zjednodušit a urychlit buildovací proces. Gu využívá balíčkovací systém pacman (převzatý z Arch Linuxu) a nástroj scratchbox2 sloužící k zjednodušení cross-kompilace. Tyto 2 nástroje jsou skryty v konzolovém přikazu „gu“, kterým se celé Gu ovládá.

2.2.2 PKGBUILD

Jedná se o shellový script obsahující informace potřebné pro build jednotlivých aplikací systémem Gu. Níže je pro představu uveden vzorový script:

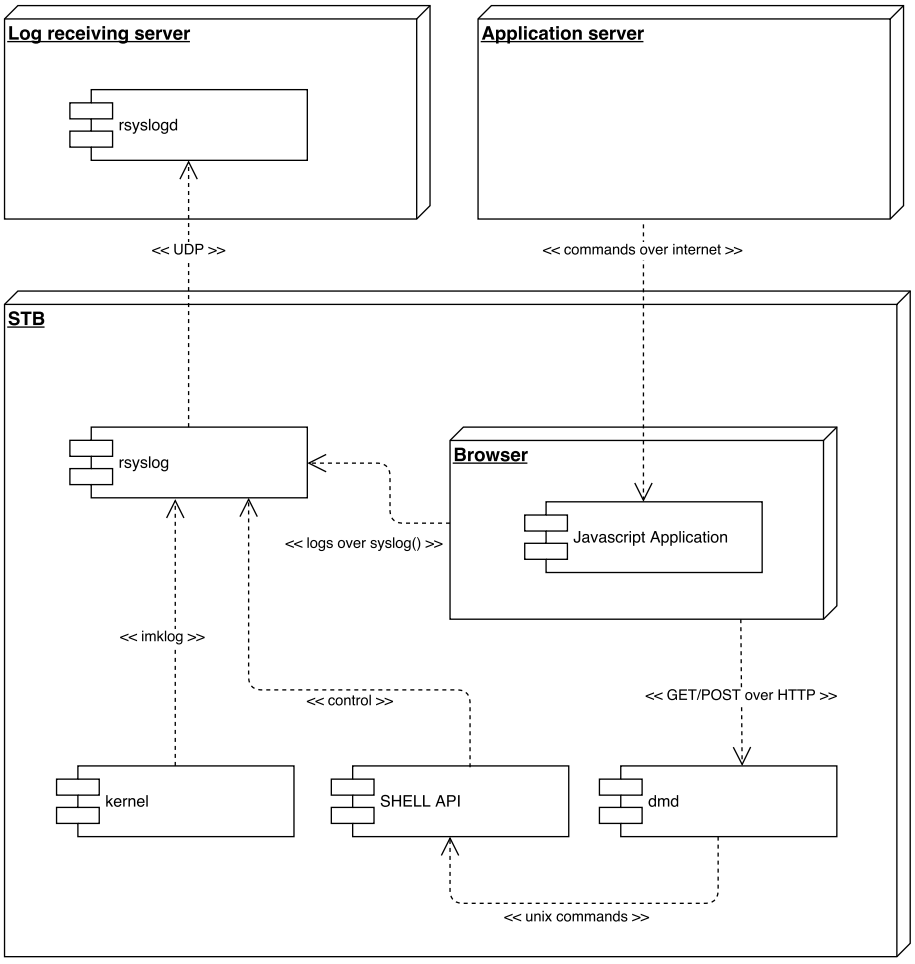
```
pkgname=libee
pkgver=0.4.1
pkgrel=1
arch=('armv7h' 'armv7sp')
depends=('libestr')

build() {
    cd ${pkgname}-${pkgver}
    PKG_CONFIG_PATH=/mnt/hdd_1/lib/pkgconfig
    ./configure --prefix=/mnt/hdd_1
    make V=1
}

package() {
    cd ${pkgname}-${pkgver}
    make install DESTDIR=${pkgdir}
}
```

Povinná pole jsou „pkgname“, označující název balíčku, „pkgver“ pro určení jeho verze, „pkgrel“ sloužící k označení verze PKGBUILD scriptu a „arch“ definující cílové architektury. Script dále obsahuje funkce zajišťující build a instalaci balíčku. Každý PKGBUILD musí obsahovat minimálne funkci package(), která nainstaluje soubory do cílové složky. Užitečnou funkcí je funkce build() sloužící k přípravě a kompilaci zdrojových kódů před samotnou jejich instalací.

2.3 Nové řešení



Obrázek 2.2: Diagram nasazení nového řešení STB

Prvně je nutno zvážit, zda problém řešit na straně serveru nebo set-top boxu. Vhodnou konfigurací logovacího démona na straně serveru, který by nepotřebné zprávy zavčas rozpoznal, zahodil a dále nezpracovával bychom splnili požadavek na snížení zátěže serverových disků. Přetížení sítě se takto vyřešit ale nedá a proto toto řešení zavrhuji. Je tedy nutno problém řešit na straně set-top boxu kde původní řešení je postaveno na busy-box syslogd. Nabízí se možnost upravit fungování tím způsobem, aby se logy s nízkou severitou už na set-top boxu zahazovaly a pouze v případě potřeby bylo umožněné na dálku změnit konfiguraci démona tak, aby se povolilo logování pro logy s nastavenou danou komponentou a severitou. To vše přes SHELL-ové API. Součástí zadání je ale i implementovat škrcení zpráv, aby nedocházelo k zahlcení linky. Takovou možnost prostý syslogd neposkytuje a je proto nutno zvážit napsání

vlastního démona či nasazení jiného, vyspělejšího logovacího démona.

2.4 Srovnání logovacích démonů

Démon v UNIXovém světě je označení pro takový proces, který oproti běžným procesům neintereaguje přímo s uživatelem, ale běží na pozadí operačního systému a funguje samostatně. Účelem logovacího démona je sběr logů od ostatních procesů, které následně v závislosti na jeho konfiguraci dokáže filtrovat a ukládat na disk či odesílat na požadovaný vzdálený server.

V této kapitole zmíním a popíši vybrané logovací démony a v závěru kapitoly je porovnám.

BusyBox Syslogd

Tato logovací utilita se skládá ze dvou démonů, jmenovitě z Klogd, který zachytává logy z kernelu a předává je ke zpracování Syslogdémonu. Syslogd pak zachytává i všechny zbylé logy a dále s nimi nakládá. Má však velice omezenou funkcionalitu. Dokáže pouze logy lokálně ukládat, přeposílat je dále po síti, zahazovat duplikáty, rotovat logy v závislosti na velikosti a filtrovat zprávy podle omezených kritérií. Dokáže totiž filtrovat pouze podle typu facility a už nikoliv podle názvu komponenty, která log vygenerovala.

Syslog-ng

Flexibilní logovací démon zaměřený na centralizované a zabezpečené logování. Má široké možnosti nastavení a poskytuje obrovské množství funkcionalit. Takže jeho vhodným nakonfigurováním se dají snadno splnit všechny vytyčené technické požadavky až na požadavek pro možnost vzdálené změny konfigurace. Je nutno ale zmínit, že pokročilé funkce jako například šifrování zpráv, bufferování nebo message-rate kontrola jsou dostupné pouze v komerční closed-source verzi.

Rsyslog

Výčet funkcionalit Rsyslogu je ještě obsáhlejší než u Syslog-ng. Technické požadavky se s jeho použitím tedy také dají splnit všechny, kromě vzdálené změny konfigurace. Oproti Syslog-ng je Rsyslog kompletně zdarma a open-source. Navíc není jen logovacím démonem, ale i analyzérem logů. Dokáže logy podle obsahu zprávy měnit, třídit a jinak s nimy nakládat. Že je Rsyslog vyspělý a kvalitní program dokazuje fakt, že je defaultním logovacím démonem na spoustě linuxových distribucích, jmenovitě například v Ubuntu. Jeho slabiny shledávám v nedostatečné dokumentaci a ve specifických případech v neefektivním analyzování logů mající za následek (obvzláště na embedded zařízení s pomalým ARM procesorem) rychlostní deficit. Jeho vývoj obstarává

z velké většiny pouze jeden člověk, jeho původní tvůrce Rainer Gerhards. A v jednom člověku není snadné dovést tak rozsáhlý projekt k dokonalosti.

Porovnání výše zmíněných logovacích utilit

Pouhým nasazením jakéhokoli známého logovacího démonu není možné splnit všechny vytyčené technické požadavky. V případě ponechání původního BusyBox syslogd démonu by pro splnění technických požadavků bylo nutno doimplementovat tolik funkcionalit, že by to výrazně přesahovalo rozsah bakalářské práce. Výhodněji se jeví nasadit pokročilý logovací démon jako je Syslog-ng či Rsyslog. Oba totiž poskytují námi požadované funkcionality. Syslog-ng však většinu z nich poskytuje pouze v placené closed-source verzi a proto jsem se rozhodl pro Rsyslog.

2.5 Vzdálená konfigurace

Rsyslog při svém zapnutí čte konfigurační soubor rsyslog.conf, který za jeho běhu není možné měnit. Je proto nutné napsat SHELL-ové API, které umožní na dálku přenastavit tento konfigurační soubor a restartovat Rsyslog. Rsyslog dokáže načíst nový konfigurační soubor zdánlivě i bez restartu a to po obdržení SIGHUP signálu. Avšak ve skutečnosti [2] k restartu dojde i tak a nelze se proto jeho restartování vyhnout.

Požadavkem zadavatele je umožnit změnu nastavení maximální povolené severity jednotlivých komponent. Tedy nastavit Rsyslog tak, aby v případě rozpoznání zprávy od určité komponenty spolu s nastavenou severitou vyšší, než je povolená, danou zprávu zahodil. Změny tímto skriptem způsobebé musí být zachovány i po restartu STB.

V této kapitole diskutuji možná řešení SHELLL-ového API. Samotná implementace je k nalezení v kapitole Realizace.

2.5.1 Logika skriptu

V úvahu připadaly 2 různé přístupy. Je možné při každém zavolání scriptu generovat celý nový konfigurační soubor a nebo parsovat stávající soubor a jeho část pomocí skriptu měnit. Výhodou prvního způsobu je, že nehrozí nechtěné přepsání jiných čásí skriptu, než bylo zamýšleno. Nevýhodou je složitější implementace z důvodu nutnosti pří generování nového skriptu zohledňovat předchozí nastavení skritpu. Tedy by bylo nutno z původního skriptu extrahovat nastavení jednotlivých komponent a to zkombinovat s novým nastavením. Rozhodl jsem se proto pro 2. způsob řešení s tím, že zadavatel bude poučen o nutnosti dodržovat určité zásady při měnění konfiguračního souboru, aby nemohlo dojít k neočekávanému chování.

2.5.2 Rozhraní skriptu

První možností je napsat API ve formě scriptu, který přijímá jednotlivé názvy komponent a jim příslušící maximální povolenou severitu jako parametry.

```
set_log_verbosity.sh [component] [severity] [component2] [severity] ...
```

Jako druhá možnost připadá v úvahu skript, který čte seznam jednotlivých komponent a maximálních povolených severit ze souboru, který má následující formát:

```
/etc/logging.conf

component1  = DEBUG
componentXY = INFO
...
DEFAULT     = INFO
```

Se zadavatelem jsme se shodli, že se lépe jeví první způsob pro jeho jednoduší a rycheljší obsluhu a navíc s jeho použitím odpadá starost o další konfigurační soubor.

2.6 Postprocessign zpráv

Všechna níže zmíněná pravidla jsou myšlena jako vzory pro zadavatele, který podle nich v budoucnu vytvoří svá vlastní.

2.6.1 Smazání podřetězce z těla syslog zprávy

Zprávy s nastavenou severitou INFO a komponentou sld_br je třeba změnit podle následujícího vzoru.

Originální zpráva:

```
2016-02-18T14:05:24+01:00 cc-b8-f1-00-6f-07 sld_br: id=559
:[stbhal.cpp:debug:520]: INFO: [94mDEBUG: InformationService:
 Reading 'nangu.video.forcedScart': false[0m
```

Výsledná zpráva:

```
2016-02-18T14:05:24+01:00 cc-b8-f1-00-6f-07 sld_br: id=559
[94mDEBUG: InformationService: Reading 'nangu.video.forcedScart':
false[0m
```

2.6.2 Zahazování zpráv podle typu komponenty

Je třeba dokázat rozpoznat a zahodit zprávy podle zadané komponenty a obsahující určitý podřetězec.

2.6.3 Převod severit

Aplikace na STB používají pro logování jinou množinu severit, než s jakými pracuje Rsyslog. Zadavatel proto požaduje změnit severity zpráv podle následující tabulky.

Tabulka 2.1: Převodní tabulka

Portal	Syslog
ERROR	ERR
WARN	WARN
INFO	NOTICE
DEBUG	INFO
TRACE	DEBUG

2.7 Rate-limiting zpráv

Původní řešení obsahuje rate-limiting implementovaný tím způsobem, že do zdrojového kódu BusyBox syslogd, který je v jednom malém C souboru, byl doimplementován statický buffer s omezenou velikostí pro 128 kB zpráv.

V Rsyslogu se podobného efektu dá dosáhnout prostým nastavením modulu imuxsock, kde parametry RateLimit.Interval a RateLimit.Burst určují maximální povolený počet zpráv za daný počet sekund.

```
module(load="imuxsock"            #Provides support for local system logging.
SysSock.RateLimit.Interval="2" #Specifies the rate-limiting interval in seconds.
SysSock.RateLimit.Burst="500"  #Specifies the rate-limiting burst in number of messages.
)
```

Původní řešení trápí neduh, kdy k vyprázdnění bufferu dochází až s nově příchozí zprávou, což má za následek, že v případě zaplněného bufferu a následující delší odmlky nově příchozích zpráv může dojit k velkému zpoždění odeslání zpráv v ten moment uložených v bufferu. Tento problém se sofistikovanějšího Rsyslogu netýká.

V sekci Testování se zabývám hledáním ideálního nastavení těchto parametrů.

Kapitola 3

Realizace

3.1 Rsyslog

3.1.1 build

Rsyslog bylo nutno zkompilovat ze zdrojových kódů včetně mnoha závislostí.

3.1.2 konfigurace

TODO jednotlivy configurace pro pravidla, dodat ukázky configů.. krome popsani rsyslog.conf i logrotate etc all scripty nebo mozna do jine podsekce toto

3.1.3 modul mmstrdel

3.1.4 modul mmsevbalbab

3.2 Shell API

Na STB je přítomna pouze minimalistická BusyBox implementace SHELLu nesoucí název ASH, která neposkytuje tak elegantní syntaxi a neposkytuje tolik rozšířených funkcionalit jako moderní SHELL-ové jazyky typu BASH. I přesto se s pomocí ASHe a základních UNIXových programů jako například GREP povedlo API naprogramovat. Skript na vstupu očekává sudý počet parametrů, kde každý sudý parametr je název komponenty a liché parametry slouží pro definování severit. Skript vyhledá v konfiguračním souboru rsyslog.conf řádek s danou komponentou a přenastaví maximální povolenou severitu. Skript kontrolojue správnost vstupních parametrů pro zabránení neočekávaného chování programu.

TODO code

3.3 dmd

Kapitola 4

Testování

Závěr

Literatura

[1] *EKT DID7006*. Dostupné z: http://exploredoc.com/doc/3174828/model-did7006-high-definition-ott-stb

[2] Gergards, R.: New rsyslog HUP processing. 2008. Dostupné z: http://blog.gerhards.net/2008/10/new-rsyslog-hup-processing.html

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

ASH Almquist Shell

dmd Download manager daemon

STB Set-top box

SIGHUP Signal Hang Up

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `BP_Vavricka_David_2016.pdf` ..... text práce ve formátu PDF