

Sem vložte zadání Vaší práce.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
KATEDRA SOFTWAREVÉHO INŽENÝRSTVÍ



Bakalářská práce

Flexibilní logování pro embedded Linuxové systémy

David Vavříčka

Vedoucí práce: Ing. Matěj Laitl

28. dubna 2016

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstráňte 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 28. dubna 2016

.....

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2016 David Vavříčka. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Vavříčka, 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

Úvod	1
1 Technické požadavky	3
1.1 Základní technické požadavky	3
1.2 Rozšířené technické požadavky	4
2 Logování v linuxových systémech	5
2.1 Syslog démon	5
2.2 Syslog formát zpráv	5
2.3 Syslog protokol	6
3 Analýza a návrh	9
3.1 Současné řešení	10
3.2 Vývoj pro STB	11
3.3 Nové řešení	12
3.4 Srovnání logovacích démonů	13
3.5 Vzdálená konfigurace	14
3.6 Postprocessign zpráv	15
3.7 Rate-limiting zpráv	16
4 Realizace	17
4.1 Rsyslog	17
4.2 Shell API	18
5 Testování	19
Závěr	21
Literatura	23
A Seznam použitých zkratek	25

Seznam obrázků

2.1	Formát syslog zprávy	5
3.1	Diagram nasazení původního řešení	9
3.2	Diagram nasazení nového řešení	12

Seznam tabulek

2.1	Tabulka severit podle RFC 5424	7
2.2	Tabulka facilit podle RFC 5424	8
3.1	Převodní tabulka severit	16

Úvod

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 zadavatel.

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 odesílaných 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 pravidla musí být možné definovat a měnit bez nutnosti nového sestavení softwaru. Řešitel vytvoří pro ukázkou 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 Javascript aplikaci řídit konfiguraci logování (viz bod Vzdálená konfigurace).

Logování v linuxových systémech

Systémoví administrátoři se musí umět vypořádat s množstvím nejrůznějších zpráv pocházejících ať už přímo z daného systému, od jednotlivých systémových komponent nebo dokonce i z jiných vzdálených systémů. Pro jejich snazší správu slouží na UNIX-ových a tedy i LINUX-ových systémech tzv. Syslog. Tím je myšleno Syslog API skrze které se odesílají jednotlivé zprávy na standardizovaný socket /dev/log. Dále je tím myšlen Syslog démon spravující jednotlivé logy. A konečně syslog protokol, tedy specifikace určující formát zpráv nebo také například způsob transportu zpráv po síti.

2.1 Syslog démon

Démon je v UNIXovém světě označení pro takový proces, který oproti běžným procesům neinteraguje přímo s uživatelem, ale běží na pozadí operačního systému a funguje samostatně. Hlavním účelem syslog 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.

2.2 Syslog formát zprávy



Obrázek 2.1: Formát syslog zprávy

Syslog zpráva se skládá ze 3 částí jak je pro lepší představu graficky vyobrazeno na obrázku výše.

V části zvané PRI (priorita) se skrývají dvě číselné hodnoty facility a severity. Facility určuje zdroj logů a severity jejich důležitost (tedy zda se jedná například pouze o debug hlášku a nebo naopak o důležitou zprávu například o pádu programu). Na základě těchto dvou hodnot syslog démon provádí základní filtraci zpráv. Se severitou a facilitou zpráv se budeme v této práci častokrát setkávat a proto jsou na konci kapitoly umístěny tabulky jejich hodnot.

V části zvané HEADER (hlavička) nalezneme časovou značku a hostname nebo IP adresu zdrojového zařízení dané zprávy. Část MSG již obsahuje text samotné zprávy.

Tyto zprávy obvykle po síti putují skrze UDP protokol, jelikož obvykle upřednostňujeme rychlost nad spolehlivostí. Syslog standard omezuje velikost paketu zprávy na maximálně 1024 Bytů.

2.3 Syslog protokol

Syslog protokol [2] spatřil světlo světa již v roce 1980. Má ho na svědomí Eric Allman, který ho psal s úmyslem využít ho pro potřeby Sendmail projektu. Nicméně postupem času byl tento protokol díky skvělému návrhu a díky své jednoduchosti adaptován i jinými projekty. Rozšířil se na různé OS a později se stal standardem pro logování na většině Unixových systémů. Tento protokol poskytuje možnost zařízením posílat notificační zprávy (logy) ať už skrze síť a nebo lokálně v rámci jednoho zařízení na syslog server, kterým je již v minulé kapitole zmíněný syslog démon.

2.3.1 The BSD syslog Protocol - RFC 3164

Až po přibližně 20-ti letech od vzniku syslog protokolu sepsal Chris Lonvick dokument RFC3164 [3] popisující daný protokol. Do té doby existovala velká spousta v detailech se lišících implementací, což mělo za následek v jistých případech nekompatibilitu. Dokument čítá celkem 19 stránek kde je především detailně popsán formát a parametry zpráv.

2.3.2 The Syslog Protocol - RFC 5424

Oficiální standard vznikl až s příchodem Syslog protokolu RFC 5424, který světu představil v roce 2009 jeho autor Rainer Gerhards. Počet stránek se rozrostl na 37 a došlo k některým zásadním a velice užitečným změnám.

Nový standard umožňuje použití libovolného transportního protokolu [4], oproti RFC 3164, kde je jako transportní protokol pevně určeno UDP [5]. Mírných změn doznal také formát zpráv, který je možno strukturovaně rozšiřovat dle specifických potřeb [6].

V této práci je vždy při zmínce o syslog protokolu myšlen standard RFC 5424.

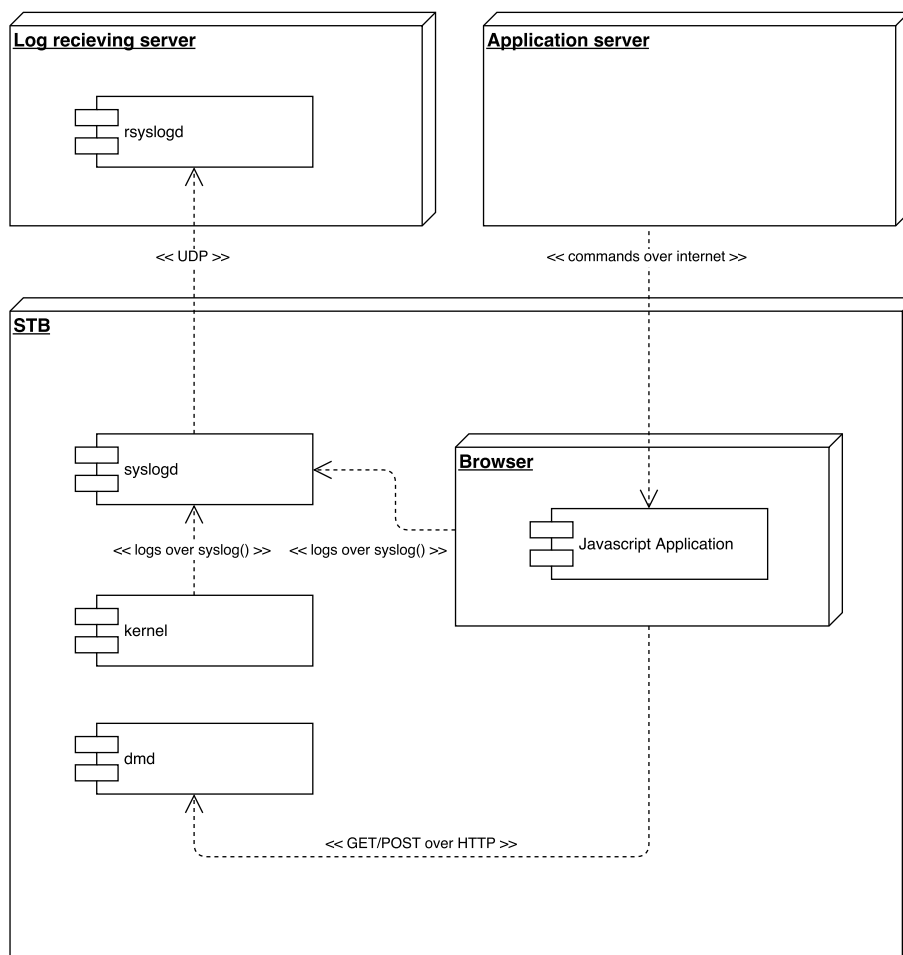
Tabulka 2.1: Seznam severit dle RFC 5424. [7]

Číselný kód	Severity	Význam
0	Emergency	System is unusable.
1	Alert	Action must be taken immediately.
2	Critical	Critical conditions.
3	Error	Error conditions.
4	Warning	Warning conditions.
5	Notice	Normal but significant condition.
6	Informational	Informational messages.
7	Debug	Debug-level messages.

Tabulka 2.2: Seznam facilit dle RFC 5424. [7]

Číselný kód	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	security/authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon
16	local use 0
17	local use 1
18	local use 2
19	local use 3
20	local use 4
21	local use 5
22	local use 6
23	local use 7

Analýza a návrh



Obrázek 3.1: Diagram nasazení původního řešení

3.1 Současné řešení

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

3.1.1 HW specifikace STB EKT DID7006

CPU: ARM 9 Ali M3733

GPU: Mali 400

RAM: 512MB DDR3

Perzistentní paměť: 512MB NAND Flash

3.1.2 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.

3.1.2.1 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.

3.1.2.2 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.

3.1.2.3 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“.

3.1.3 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.

3.1.4 Servery pro sběr logů

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

3.2 Vývoj pro STB

3.2.1 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 příkazu „gu“, kterým se celé Gu ovládá.

3.2.2 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
}

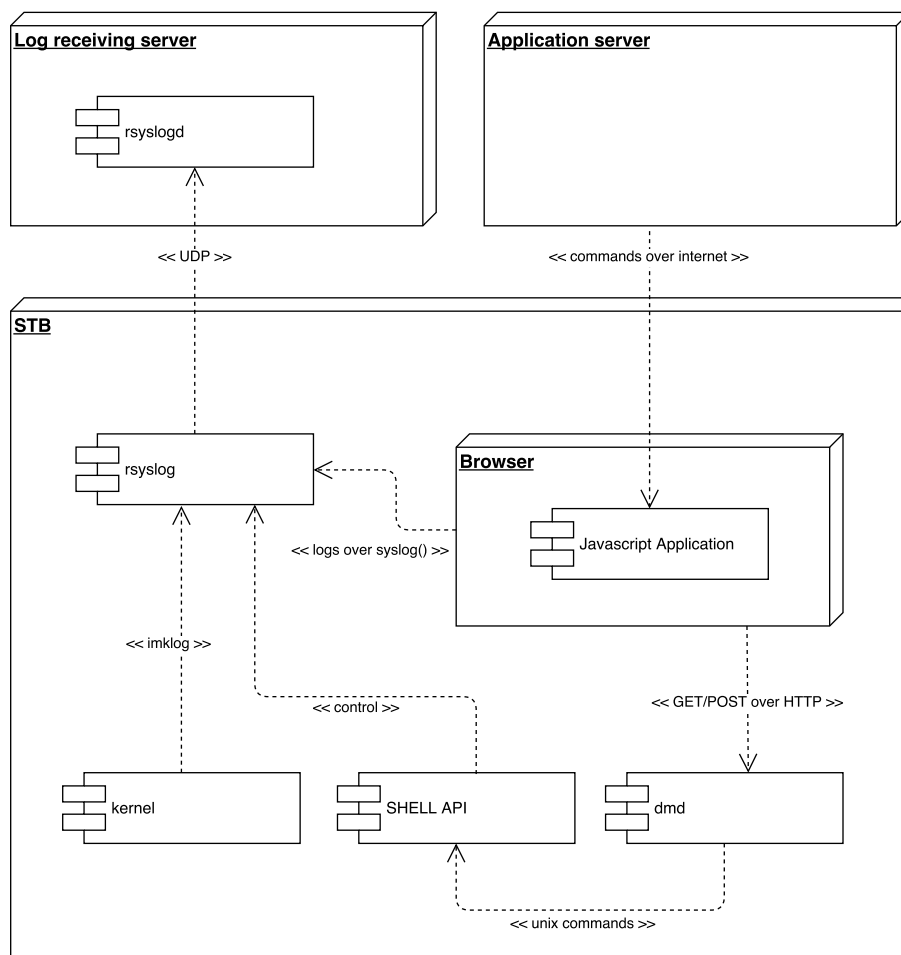
package() {
    cd ${pkgname}-${pkgver}
```

3. ANALÝZA A NÁVRH

```
    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 minimálně funkci package(), která nainstaluje soubory do cílové složky. Užitečnou funkcí je funkce build() sloužící k přípravě a kompilaci zdrojových kódů před samotnou jejich instalací.

3.3 Nové řešení



Obrázek 3.2: Diagram nasazení nového řešení

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.

3.4 Srovnání logovacích démonů

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

3.4.1 BusyBox Syslogd

Tato logovací utilita se skládá ze dvou démonů, jmenovitě z Klogd, který zachytává logy z kernelu a předává je ke zpracování Syslogdemonu. 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.

3.4.2 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.

3.4.3 Rsyslog

Výčet funkcionalit Rsyslogu je ještě obsáhlejší než u Syslog-ng [8]. 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 nimi 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 (obzvláště 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.

3.4.4 Porovnání výše zmíněných logovacích utilit

Pouhým nasazením jakéhokoli známého logovacího démonu není možné splnit všechny vytyčené technické požadavky. V případě ponechání původního Busy-Box 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.

3.5 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 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 [9] 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 způsobené musí být zachovány i po restartu STB.

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

3.5.1 Logika skriptu

V úvahu připadaly 2 různé přístupy. Je možné při každém zavolání skriptu 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 částí skriptu, než bylo zamýšleno. Nevýhodou je složitější

implementace z důvodu nutnosti při generování nového skriptu zohledňovat předchozí nastavení skriptu. 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.

3.5.2 Rozhraní skriptu

První možností je napsat API ve formě skriptu, 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 zjednodušení a rychlejší obsluhu a navíc s jeho použitím pro zadavatele odpadá starost o další konfigurační soubor.

3.6 Postprocessign zpráv

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

3.6.1 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
```

3.6.2 Zahazování zpráv podle typu komponenty

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

3.6.3 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 3.1: Převodní tabulka severit

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

3.7 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 dojít 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.

Realizace

4.1 Rsyslog

4.1.1 Build

Jelikož Rsyslog není součástí SDK, bylo ho nutné a všechny knihovny na kterých závisí (zlib, libestr, libee, liblogging, libfastjson) zkompileovat a posléze nainstalovat na STB.

4.1.2 Konfigurace

TODO jednotlivy konfigurace pro pravidla, dodat ukázky configů.. krome popsaní rsyslog.conf i logrotate etc all scripty nebo možná do jiné podsektce toto

4.1.3 spouštění

protože se musí upravit rsys.conf dle se nastaví MAC addr která následně bude v hlavičce každé zprávy

4.1.4 Programování vlastních modulů

<http://www.rsyslog.com/doc/v8-stable/configuration/modules/index.html> TODO popsat vyvoj modulu, typy modulu viz odkaz nahore

4.1.4.1 Modul mmstrdel

```
module(load="mmdelstr")  
action(type="mmdelstr" stringtobedeleted="Some string")
```

4.1.4.2 Modul mmsevrewrite

```
module(load="mmsevrewrite")  
action(type="mmsevrewrite" severity="debug")
```

4.2 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 kontroluje správnost vstupních parametrů pro zabránění neočekávaného chování programu.

Testování

Závěr

Literatura

- [1] EKT DID7006. *ExploreDoc* [online]. 2015 [cit. 2016-04-26]. Dostupné z: <http://exploredoc.com/doc/3174828/model-did7006-high-definition-ott-stb>
- [2] Deveriya, A.: An Overview of the syslog Protocol. *Network Administrators Survival Guide*, 2005: s. 181–184, ISBN: 978-1-58705-211-8.
- [3] Lonvick, C.: The BSD syslog Protocol. *IETF Tools Pages* [online]. 2001 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc3164>
- [4] Gerhards, R.: Transport Layer Protocol. *IETF Tools Pages* [online]. 2009 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc5424#page-7>
- [5] Lonvick, C.: Transport Layer Protocol. *IETF Tools Pages* [online]. 2001 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc3164#page-5>
- [6] Gerhards, R.: Syslog Message Format. *IETF Tools Pages* [online]. 2009 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc5424#page-8>
- [7] Gerhards, R.: The Syslog Protocol. *IETF Tools Pages* [online]. 2009 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc5424>
- [8] RSyslog - Features. *RSyslog* [online] [cit. 2016-04-27]. Dostupné z: <http://www.rsyslog.com/doc/features.html>
- [9] Gergards, R.: New rsyslog HUP processing. *Rainer's Blog* [online]. 2008 [cit. 2016-04-26]. Dostupné z: <http://blog.gerhards.net/2008/10/new-rsyslog-hup-processing.html>

Seznam použitých zkratk

API Application Programming Interface

ASH Almquist Shell

CPU Central Processing Unit

dmd Download manager daemon

GPU Graphics processing unit

OS Operační systém

RAM Random Access memory

RFC Request for Comments

SDK Software Development Kit

STB Set-top box

SIGHUP Signal Hang Up

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 L ^A T _E X
	text	text práce
	thesis.pdf	text práce ve formátu PDF
	BP_Vavricka_David_2016.pdf	text práce ve formátu PDF