

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

2. května 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 2. května 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

<b>Úvod</b>	<b>1</b>
<b>1 Technické požadavky</b>	<b>3</b>
1.1 Základní technické požadavky . . . . .	3
1.2 Rozšířené technické požadavky . . . . .	4
<b>2 Logování v linuxových systémech</b>	<b>5</b>
2.1 Syslog démon . . . . .	5
2.2 Syslog formát zpráv . . . . .	5
2.3 Syslog protokol . . . . .	6
<b>3 Analýza a návrh</b>	<b>9</b>
3.1 Současné řešení . . . . .	10
3.2 Nové řešení . . . . .	11
3.3 Srovnání logovacích démonů . . . . .	11
<b>4 Realizace</b>	<b>15</b>
4.1 Sestavení a instalace Rsyslogu na STB . . . . .	15
4.2 Konfigurace Rsyslogu . . . . .	17
4.3 Rsyslog moduly . . . . .	22
4.4 Vzdálená konfigurace . . . . .	25
<b>5 Testování</b>	<b>27</b>
<b>Závěr</b>	<b>29</b>
<b>Literatura</b>	<b>31</b>
<b>A Seznam použitých zkratk</b>	<b>33</b>
<b>B Obsah příloženého CD</b>	<b>35</b>



---

## Seznam obrázků

2.1	Formát syslog zprávy . . . . .	5
3.1	Diagram nasazení původního řešení . . . . .	9
3.2	Diagram nasazení nového řešení . . . . .	12
4.1	Rsyslog queues . . . . .	19



---

## Seznam tabulek

2.1	Tabulka severit podle RFC 5424 . . . . .	7
2.2	Tabulka facilit podle RFC 5424 . . . . .	7
4.1	Převodní tabulka severit . . . . .	24





---

# 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 maximální severity zpráv pro jednotlivé komponenty a dále toto API musí umožnit nastavit výchozí severity, 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 filtrační pravidla.

## 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 zmiňce 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.

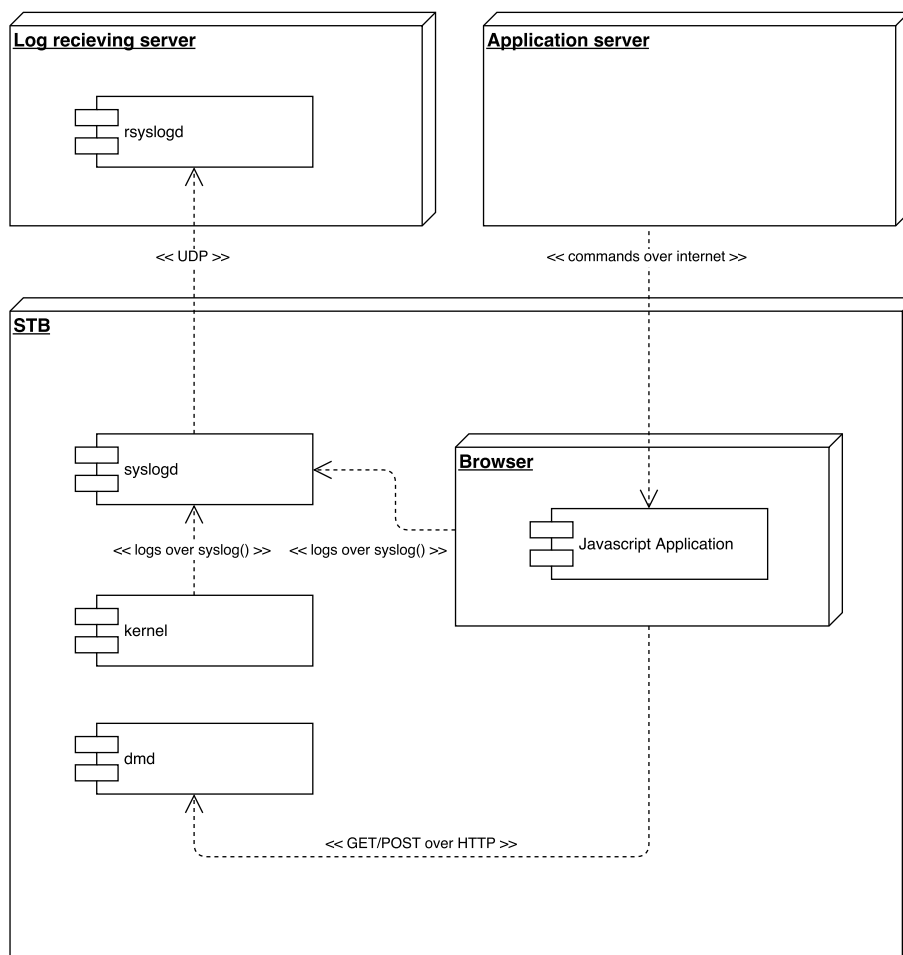
Tabulka 2.2: Seznam facilit dle RFC 5424. [7]

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
...	...
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.

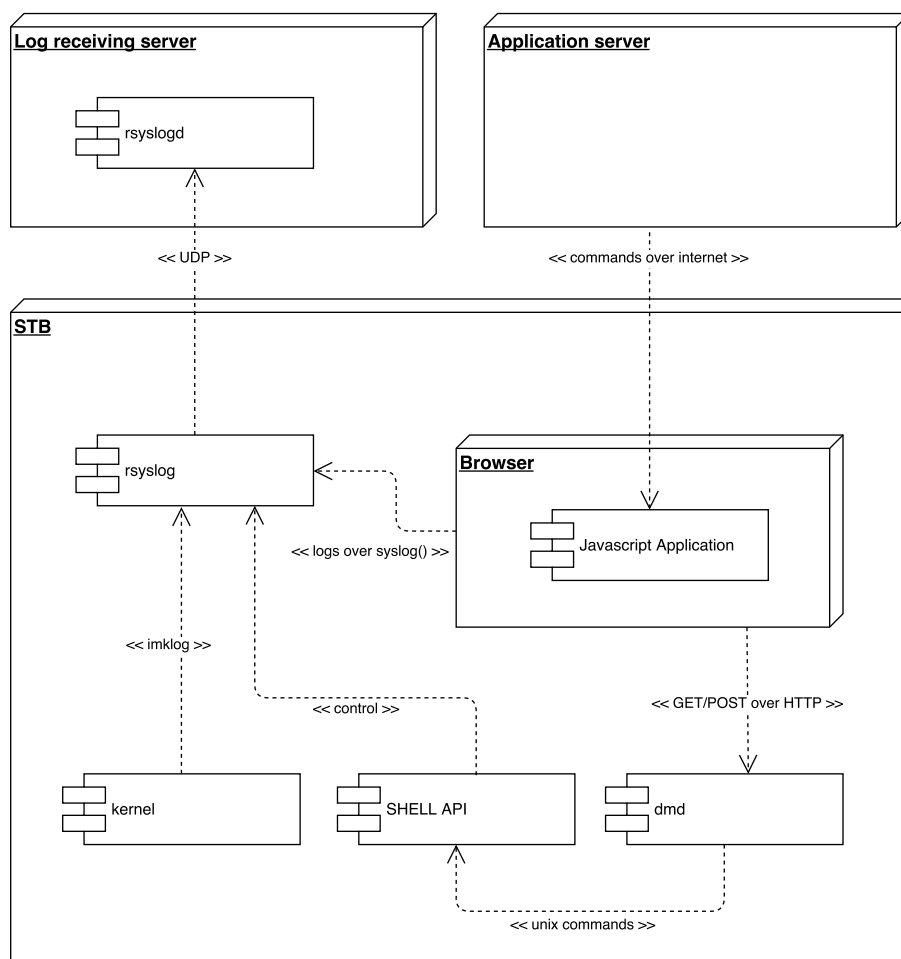
## 3.2 Nové řešení

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.

## 3.3 Srovnání logovacích démonů

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

### 3. ANALÝZA A NÁVRH



Obrázek 3.2: Diagram nasazení nového řešení

#### 3.3.1 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.

#### 3.3.2 Syslog-ng

Flexibilní logovací démon zaměřený na centralizované a zabezpečené logování. Má široké možnosti nastavení a poskytuje obrovské množství funkcio-

nalit. 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.

#### 3.3.3 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.

#### 3.3.4 Porovnání 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.

#### 3.3.5 RELP

TODO

#### 3.3.6 Šifrování zpráv

TODO

#### 3.3.7 Komprese zpráv

TODO



## Realizace

### 4.1 Sestavení a instalace Rsyslogu na STB

Rsyslog není součástí SDK a tak jej bylo nutné a všechny knihovny na kterých závisí (zlib, libestr, libee, liblogging a libfastjson) zkompileovat a posléze nainstalovat na STB. K tomu bude použit systém Gu.

#### 4.1.1 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.

#### 4.1.2 PKGBUILD

Jedná se o shellový script obsahující informace potřebné pro sestavení jednotlivých aplikací systémem Gu. Jeho syntaxe je podrobně popsána v odstavci níže.

#### 4.1.3 Sestavení a instalace aplikací

Jako příklad zde uvádím soubor PKGBUILD pro sestavení a instalaci Rsyslogu. V podobném duchu jsou napsány i skripty pro ostatní aplikace a knihovny.

```
pkgname=rsyslog
pkgver=8.16.0-nangu-0.3
arch=('armv7h' 'armv7sp')
depends=('zlib' 'libestr' 'libee' \
'liblogging' 'libfastjson')
source=${pkgname}-${pkgver}.tar.gz
```

#### 4. REALIZACE

---

```
md5sums=( 'SKIP' )

build() {
    cd ${pkgname}
    PKG_CONFIG_PATH=/mnt/hdd_1/lib/pkgconfig
    autoreconf -fvi
    ./configure --prefix=/mnt/hdd_1 \
        --disable-uuid --enable-mmsequence \
        --enable-mmsevrewrite --enable-mmdelstr
    make V=1
}

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

**pkgname:** Název balíčku (měl by se shodovat s názvem zdrojového archivu aplikace).

**pkgver:** Verze balíčku (měla by se shodovat s verzí sestavované aplikace).

**arch:** Pole specifikující architektury cílových systémů.

**depends:** Pole názvů balíčků, které musí být nainstalovány před spuštěním tohoto softwaru.

**source:** Seznam souborů nutných pro sestavení balíčku. Soubory mohou odkazovat na lokální úložiště, nebo na vzdálený server, v tom případě se skript postará o jejich stažení. Komprimované soubory skript automaticky rozbalí.

**md5sum:** Pole kontrolních součtů pro každý specifikovaný „source“.

**build():** Nepovinná funkce obsahující příkazy vedoucí ke konfiguraci a sestavení aplikace.

**package():** Povinná funkce instalující soubory. Funkce je spouštěna až po exekuci ostatních nepovinných funkcí.

**pkgdir:** Proměnná obsahující umístění kořenového adresáře instalované aplikace.

Po vstoupení do adresáře obsahujícím zdrojové kódy aplikace a skript PKGBUILD stačí zadat příkaz „gu build“, čímž se uvede do chodu celý skript. Nejdříve se inicializují proměnné, posléze se vykoná funkce build() a nakonec package(), čímž by měl (pokud nedojde k žádné chybě) vzniknout v umístění „pkgdir“ balíček s přeloženou aplikací. Příkazem „gu install <název-vzniklého-balíčku.tar.xz>“ nainstalujeme balíček do SDK.



## 4.2 Konfigurace Rsyslogu

Rsyslog se řídí podle pravidel specifikovaných v konfiguračním souboru `/etc/rsyslog.conf`. Za použití tzv. compatibility módu [9] podporuje syntaxi známou z dříve na linuxu hojně užívaného démonu `syslogd`. Níže je uveden příklad konfiguračního souboru `syslog.conf` [10] pro `syslogd`. Na levo je možno specifikovat kritéria pro filtraci zpráv podle jejich severity a facility. Na pravo je pak umístění, kam zprávy odesílat.

```
*.=crit;kern.none           /var/adm/critical
kern.*                       /var/adm/kernel
kern.crit                    @finlandia
kern.crit                     /dev/console
kern.info;kern.!err          /var/adm/kernel-info
```

Díky této podpoře je možno velice snadno migrovat na Rsyslogd. Pro využití i jiných než základních funkcí Rsyslogu, je nutno k nakonfigurování pravidel použít syntaxi jazyku Rainerscript.

### 4.2.1 Rainerscript

Rainerscript [11] je skriptovací jazyk navržený ke správě síťových událostí (převážně nakládání se syslog zprávami) a dále ke konfiguraci softwaru, pro který je používán. Tento jazyk je totiž teoreticky použitelný v různých typech softwarů, nicméně v době psaní této práce byl vyvíjen a reálně používán pouze v Rsyslogu.

Jedná se o netypový jazyk, který poskytuje podporu regulárních výrazů [12], zná všechny běžné operátory pro práci s číselnými nebo textovými proměnnými jako např. `==`, `!=`, `<>`, `<`, `>`, `<=`, `>=`, `+`, `-`, `*`, `/`, `%` a podporuje užitečné funkce především pro práci s textovými řetězci. Podrobnosti jsou k nalezení v dokumentu s formální definicí jazyka [13].

### 4.2.2 rsyslog.conf

Níže je uvedena ukázka RainerScript konfiguračního souboru, kde jsou předvedeny základní konstrukce používané při jeho tvorbě.

```
module(load="NazevModulu")
```

Na začátku souboru je možné načíst různé moduly, poskytující doplňující funkcionality. Příkladem může být modul `imklog` čtoucí zprávy z kernelu nebo `omfwd` umožňující posílat zprávy zkrze UDP nebo TCP protokoly. Je nutno poznamenat, že některé moduly nejsou defaultně kompilovány spolu s rsyslogem a je proto nutno přidat příkazu `compile` parametr `--enable-<název_modulu>`.

```
set $!Promenna="hodnota";
unset $!Promenna;
```

#### 4. REALIZACE

---

Pomocí klíčových slov `set` a `unset` je možné deklarovat a rušit proměnné. Ty začínají znaky `$!` a příkaz je nutno ukončit středníkem.

```
template(name="nazev-sablony" type="string"
         string="<%pri%>%TIMESTAMP:::date-rfc3164%
         %$!macaddr% %syslogtag% id=%$!counter%%msg%\n"
         )
```

Šablony definují podobu zpráv. Do proměnné `string` lze vložit libovolné textové řetězce. Proměnná mezi dvojicí znaků procento je nahrazena jejich obsahem. Použity mohou být vlastní lokální proměnné a nebo tzv. replacement proměnné, vztahující se typicky k syslog zprávě, ve kterých je uložena např. severita nebo například čas odeslání. Nejdůležitější je proměnná `msg`, kde je uložen text zprávy.

```
action(template="nazev-sablony" type="omfwd"
        Target="192.168.1.10" Port="5514" Protocol="udp" )
```

Pomocí akcí se volají jednotlivé moduly. Ty mohou mít různé parametry, které upravují jeho chování. V ukázce výše se volá modul `omfwd` sloužící k posílání zpráv skrze síť. Parametry specifikují konkrétní IP a port sítě, typ protokolu a název šablony, která upravuje formát zpráv.

```
if $programname == 'solid' and $syslogseverity > 5
then {
    #nejaka akce
    stop
}
```

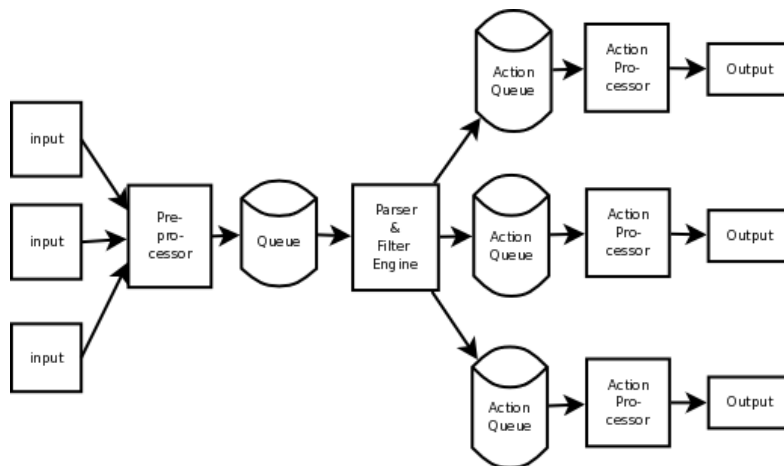
Použití podmínky `if` je zřejmé z ukázky výše. V podmínce `if` se obvykle volají různé akce.

Veškerý text za symbolem „`#`“ až do konce řádky je považován za komentář.

Nyní po obeznámení se základní syntaxí Rainerscriptu je možno popsat fungování zachytávání jednotlivých zpráv. Ty imaginárně putují skrze kód `rsyslog.conf` odshora dolů, kde prochází skrze podmínky a akce (které je eventuálně upravují nebo např. zapisují do souborů) a to až do doby, než narazí na klíčové slovo `stop` nebo na poslední řádek souboru.

#### Rsyslog queues

Princip fungování front v Rsyslogu je znázorněn na obrázku níže. Zprávy (ať už lokální nebo ze vzdáleného zdroje) putující do Rsyslogu např. skrze UDP) vstupují do Rsyslogu na obrázku zleva, kde jsou předzpracovány a putují do Main Queue. V dalším kroku jsou zprávy tříděny podle pravidel specifikovaných v `rsyslog.conf` a putují do jednotlivých action queues. Následně proběhne



Obrázek 4.1: Rsyslog queues [14]

finální zpracování zpráv a opouštějí rsyslog skrze výstupní modul ať už zápisem do lokálního souboru nebo např. předáním jinému procesu.

Všechny zprávy tedy projdou nejprve skrze Main Queue a poté skrze libovolný počet Action Queues. Je nutno zmínit, že každá akce obsahuje vlastní frontu, ve které zprávy čekají na zpracování. V případě, že v akci žádnou frontu nepotřebujeme, stačí nastavit jako Action Queue mód Direct mode.

Existují totiž čtyři módy front. Speciálním (a výchozím) je Direct mode. Direct Queues ve skutečnosti nejsou frontami. Neobsahují totiž žádný buffer a zprávy rovnou přeposílají dál. Vhodné jsou například při zápisu logů do souboru na lokální úložiště.

Druhým typem jsou In-Memory Queues, které udržují zprávy v na operční paměti uloženém bufferu. Výhodou je vysoká rychlost a nevýhodou zase hrozba ztráty dat v případě neočekávaného pádu systému. Existují dva podtypy In-Memory front. Můžou být implementovány formou spojového seznamu (LinkedList queue) nebo za použití pole ukazatelů na prvky fronty (FixedArray) s předem pevně určeným počtem prvků. FixedArray mode je nejrychlejší ze všech módů a je vhodný pro případy, kdy očekáváme pouze malý počet prvků ve frontě. LinkedList queue dynamicky alokuje místo v paměti pro každou zprávu. Obecně se doporučuje použití implementace postavené na spojovém seznamu, jelikož výkonostní propad oproti fixnímu poly je zanedbatelný, zato úspora paměťového místa může být značná.

Třetí typ front představují Disk Queues, které jak z názvu vyplývá bufferují na disk. Využití najdou v případech kdy si žádáme vysokou soplehlivost. Jejich nevýhodou je pomalost limitovaná rychlostí perzistentních úložišť.

Disk-Assisted Memory Queues fungují stejně jako In-Memory Queues a navíc v případě potřeby (kterou může být nedostatek místa nebo nutnost uložení zpráv z bufferu z nutnosti zálohy zpráv z perzistentní paměti před vypnu-

tím systému) dokáží obsah fronty nebo její část přesunout na disk. Používají chytrý algoritmus, který při malém vytížení vůbec nepoužívá disk queues. V případě vyčerpání kapacity In-Memory fronty se algoritmus postará o přesun části zpráv do diskové fronty.

### Rate-limiting za použití Rsyslog queues

Původní řešení trápil neduh, kdy k vyprázdnění bufferu docházelo až s nově příchozí zprávou, což mělo za následek, že v případě zaplněného bufferu a následující delší odmlky nově příchozích zpráv mohlo dojít k velkému zpoždění odeslání zpráv v ten moment uložených v bufferu. Tento problém se nasazením Rsyslogu a použitím jeho sofistikovaných front vyřeší. V této kapitole je popsána implementace fronty, která má za cíl škrtit tok odchozích zpráv podle stejných kritérií, jak tomu bylo původně. Původní řešení bylo postavené na syslogd démonu doplněné o hardcodovaný rate-limiting zpráv a fungovalo následovně. Všechny příchozí zprávy putovaly do fronty o fixní velikosti 128 kB. V případě, kdy tok odchozích zpráv překročil 150 kbit/s nebo počet zpráv ve frontě přesáhl 800 byly nově příchozí zprávy zahozeny. Rsyslog neumožňuje nastavení maximálního toku sítě, avšak poskytuje možnost nastavit pro frontu tyto parametry:

**queue.size:** Maximální počet zpráv ve frontě.

**queue.dequeuebatchsize:** Maximální počet zpráv vystupujících z fronty v jeden okamžik.

**queue.dequeueslowdown:** Zpoždění (v  $\mu s$ ) mezi odesíláním jednotlivých zpráv (v případě `queue.dequeuebatchsize > 1` bloků zpráv).

Queue.size tedy nastavíme stejně na 800 zpráv. Nyní je třeba zjistit parametry pro dequeuebatchsize a dequeueslowdown. Z měření bylo zjištěno, že průměrná velikost UDP paketu se zprávou ze STB má 150B. Pomocí níže zobrazených vzorců jsme snadno spočetli požadované zpoždění v mikrosekundách, aby zátěž linky nepřekročila 150 kbit/s.

$$\text{počet zpráv} = \frac{\text{bandwidth}}{\text{velikost paketu}} = \frac{150 \cdot 1000 \text{ [b]}}{8 \cdot 150 \text{ [b]}} = 125$$

$$\text{zpoždění} = \frac{1 \text{ sekunda}}{\text{počet zpráv}} = \frac{1000000}{125} = 8000 [\mu s]$$

Ještě zbývá zvolit typ fronty, kde kvůli důrazu na rychlost byla zvolena implementace založená na spojovém seznamu. Výsledný kód pro akci s In-memory frontou s nastaveným rate-limitingem vypadá následovně:

```
action(type="omfwd" target="192.168.1.10" port="5514"
queue.size="800" queue.dequeueslowdown="8000"
queue.dequeuebatchsize="5" queue.type="LinkedList" )
```

### 4.2.3 Razítkování zpráv

Požadavkem zadavatele bylo značit všechny zprávy vzestupnou řadou čísel kvůli rozpoznání případného výpadku některých zpráv. Byl pro to použit modul `mmsequence` [15], který byl nastaven jako je ukázáno v kódu níže.

```
module(load="mmsequence")
action( type="mmsequence"
        from="1"
        to="1048576"    # 2^20
        var="$!counter" )
```

Proměnné „`$!counter`“ je tak přiřazeno číslo „1“ a s každou další příchozí zprávou se číslo zvýší o „1“ dokud nenarazí na maximální hodnotu specifikovanou parametrem „`to`“, kdy je číslo vyresetováno na původní hodnotu „1“. Tato proměnná je následně používána v šablonách a je tak připojena k tělu každé zprávy.

### 4.2.4 Formát logů

Požadavkem zadavatele je, aby zprávy měly identický formát, jako tomu bylo u původního řešení. Důvodem je, že logy ze STB sbírá vzdálený server, který očekává daný formát.

Vzor zprávy:

```
May  1 19:04:54 cc-b8-f1-04-17-89 solid:
id=8853 Player Time [00:06:56.96]
```

Formát:

```
<PRI><RFC3164 date> <STB mac address> <component>:
<id=NUM> <message>
```

Můžeme si povšimnout, že vzorová zpráva začíná datumem a přitom formát zprávy obsahuje na prvním místě číslo PRI a až poté následuje datum. Hodnota PRI totiž není viditelná a slouží logovacímu démonu, který z ní dokáže vypočítat severity a facility zprávy.

#### 4.2.4.1 Šablony

Šablony specifikují výslednou podobu zpráv a jejich použití je intuitivní, proto rovnou předvedu hotovou implementaci, která splňuje požavky pro formát.

#### Implementace šablony

Základní šablona pro zprávy lokálně ukládané na STB:

```
template( name="local-template" type="string"
string="%TIMESTAMP::date-rfc3164% %\%!macaddr%
%syslogtag% id=%\%!counter%%msg%\n" )
```

**name:** Název šablony, který slouží jako identifikátor pro jednotlivé akce.

**type:** Definuje typ šablony, které umožňují různé způsoby jak specifikovat obsah šablony. V této ukázce je zvolen styl string.

**string:** Tento parametr je možno definovat jen v případě zvolení typu šablony „string“. Jeho obsahem pak jsou konstantní textové řetězce a proměnné, které jsou vměstnány mezi dvojice znaků procento.

### 4.3 Rsyslog moduly

Rsyslog poskytuje podporu tzv. modulů, které mohou zásadně rozšiřovat jeho funkcionalitu. Tyto moduly jsou obvykle napsány v jednom C souboru. Pro jejich zprovoznění je třeba náležitě upravit Makefile a zkompileovat Rsyslog se zapnutou jejich podprou.

Moduly se dělí na 6 základních kategorií, zde zmíním 3 nejpoužívanější:

**Output Modules:** Výstupní moduly umožňují posílat zprávy na různé cíle.

Je tak možno implementovat například modul, který dokáže odesílat zprávy do nějaké exotické databáze, kterou Rsyslog v základu nezná. Jako příklad výstupního modulu zmíním ommail, což je modul sloužící k posílání zpráv skrze mail.

**Input Modules:** Vstupní moduly jak název napovídá umožňují přijímat zprávy z různých zdrojů. Hojně používaný je imklog sbírající zprávy z linuxového jádra nebo imudp přijímající zprávy skrze UDP protokol.

**Message Modification Modules:** Moduly, které modifikují obsah zprávy.

Jako příklad uvedu v této práci použitý modul mmsequence, který generuje podle zadaných kritérií číselné řady.

V této práci byly naprogramovány dva moduly, oba z kategorie Message Modification Modules.

#### 4.3.1 Modul mmdelstr

Tento modul slouží ke smazání zadaného podřetězce z těla zprávy.

Použití:

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

Modul má pouze jeden parametr „stringtobedeleted“, kterým se specifikuje podřetězec, který má být smazán. V případě neexistence takového podřetězce, zůstává zpráva netknutá. V případě výskytu vícero takových podřetězců, je smazán pouze první.

### 4.3.2 Modul mmsevrewrite

Mmsevrewrite dokáže měnit severitu zadané zprávy. Rsyslog v základu tuto funkcionalitu neumožňuje, protože se jedná o vzácný požadavek. Modul je použit v této práci pro konverzi zpráv pocházejících z aplikace používající s Rsyslogem jinak nekompatibilní množinu severit.

Použití:

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

Modul obsahuje pouze jeden parametr „severity“, kterým se určuje nová severita zprávy. V případě zadání neplatné severity Rsyslog zaloguje chybovou hlášku a upravovaná zpráva zůstane v původním stavu.

### 4.3.3 Postprocessing zpráv

Zadavatel si přeje vytvoření ukázek pravidel pro RainerScript, které budou sloužit jako vzor, podle kterých si sám v budoucnu vytvoří sadu vlastních pravidel.

#### 4.3.3.1 Zahazování zpráv podle typu komponenty

Pravidlo, které rozpozná zprávy podle zadané komponenty a obsahu zprávy a ty následně zahodí.

Implementace pravidla:

```
if $programname == 'solid' and \
$msg contains "Player_GetState" then
{
    stop
}
```

Pokud zpráva pochází z komponenty „solid“ a obsahuje zmíněný podřetězec, klíčové slovo stop okamžitě zahodí v ten moment zpracovávanou zprávu.

#### 4.3.3.2 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 4.1: Převodní tabulka severit

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

Implementace pravidla:

```
if $msg contains "mTRACE: " then
{
    action(type="mmsevrewrite" severity="debug")
}
else if $msg contains "mDEBUG: " then
{
    action(type="mmsevrewrite" severity="info")
}
```

Zprávy obsahující nesprávně nastavené severity je možno opravit podle výše zmíněného pravidla. Bylo totiž vyzorováno, že ony postižené zprávy obsahují podřetězec ve formátu „m<SEVERITY>: “. Stačí pak na danou zprávu zavolat akci mmsevrewrite s parametrem severity vyplněným podle převodní tabulky severit.

#### 4.3.3.3 Smazání podřetězce z těla syslog zprávy

Zprávám pocházejícím z komponenty solid je třeba smazat zadaný podřetězec.

Implementace pravidla:

```
if $programname == 'solid' and $msg contains \
":[notificationFromPlayer]: INFO: " then
{
    action(type="mmdelstr" \
    stringtobedeleted="":[notificationFromPlayer]: INFO: ")
}
```



## 4.4 Vzdálená konfigurace

Požadavkem zadavatele je umožnit změnu nastavení maximální povolené severity jednotlivých komponent. Rsyslog neumožňuje změnu konfiguračního souboru za jeho běhu a proto je nutno implementovat API, které umožní přenastavení konfiguračního souboru a jeho znovunačtení (restartováním Rsyslogu). Změny tímto skriptem způsobené musí být zachovány i po restartu STB.

### 4.4.1 Návrh API

Skript musí umět nastavit konfiguraci souboru `rsyslog.conf` tak, aby v případě rozpoznání zprávy od určité komponenty spolu s nastavenou severitou vyšší, než je povolená, danou zprávu zahodil. Musí tedy umět v daném skriptu nalézt část kódu, kde se zpracovává zadaná komponenta a tuto část kódu vhodně upravit.

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.

### 4.4.2 Výběr implementačních nástrojů

Jazyky jako Perl nebo Python se jeví jako ideální pro napsání skriptu na zpracování textu. Tyto ani jiné podobné jazyky ovšem nejsou součástí SDK a zadavatel si nepřeje jejich instalaci z důvodu omezené paměti nevalného výkonu embedded zařízení. Nezbyva než se spokojit s na STB přítomnou minimalistickou BusyBox implementací SHELLu nesoucí název ASH. Ta sice neposkytuje tak elegantní syntaxi a neposkytuje tolik rozšířených funkcionalit jako moderní SHELL-ové jazyky typu BASH, ale i přesto se s její pomocí a základních UNIXových programů jako například GREP je možno API naprogramovat.

### 4.4.3 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] ...
```

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 jednodušší a rychlejší obsluhu a navíc s jeho použitím pro zadavatele odpadá starost o další konfigurační soubor.

### 4.4.4 Implementace API

Skript na vstupu očekává sudý počet parametrů, kde každý sudý parametr je název komponenty a liché parametry slouží pro definování severit pomocí jejich číselných hodnot. 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

TODO



---

## **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] Compatibility Notes for rsyslog. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: <http://www.rsyslog.com/doc/v8-stable/compatibility/v3compatibility.html>
- [10] syslog.conf(5). *Linux man pages* [online] [cit. 2016-04-30]. Dostupné z: <http://linux.die.net/man/5/syslog.conf>

- [11] RainerScript. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: <http://www.rsyslog.com/doc/rainerscript.html>
- [12] The Property Replacer. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: [http://www.rsyslog.com/doc/v8-stable/configuration/property\\_replacer.html](http://www.rsyslog.com/doc/v8-stable/configuration/property_replacer.html)
- [13] RainerScript formal definition. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: [http://www.rsyslog.com/doc/rscript\\_abnf.html](http://www.rsyslog.com/doc/rscript_abnf.html)
- [14] Rsyslog Queues. *RSyslog* [online] [cit. 2016-05-01]. Dostupné z: [http://www.rsyslog.com/doc/v8-stable/whitepapers/queues\\_analogy.html](http://www.rsyslog.com/doc/v8-stable/whitepapers/queues_analogy.html)
- [15] Number generator and counter module. *RSyslog* [online] [cit. 2016-05-01]. Dostupné z: <http://www.rsyslog.com/doc/mmsequence.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



## 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 <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	BP_Vavricka_David_2016.pdf .....	text práce ve formátu PDF