Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Pojítko Skywire - řídící modul

Bc. Jiří Cidlina

Vedoucí práce: Ing. Jiří Chludil

27. ledna 2015

Poděkování

Rád bych poděkoval vedoucímu práce, panu Ing. Jiřímu Chludilovi a jejímu oponentovi, panu Ing. Jiřímu Melnikovi, za ochotu a pomoc při tvorbě této práce a své rodině a přátelům za jejich toleranci.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 27. ledna 2015 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Cidlina, Jiří. *Pojítko Skywire - řídící modul.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce se zabývá vzdáleným řízením zařízení s OS Linux. Postupně řeší možnosti implementace vzdáleného ovládání síťových prvků pomocí webového rozhraní a SNMP. Blíže se zaměřuje na libmicrohttpd a NetSNMP, s jejichž pomocí je řešena technologie. Ta je vztažena k zařízení na tvorbu mikrovlnných spojů, založených na platformě SH4. Realizace začíná s prostředím, které bylo používáno pro tvorbu jeho firmwaru a postupně jej upravuje a doplňuje. Původně požadované rozšíření řídícího modulu o podporu SNMP a vylepšený sběr dat postupně přechází až k jeho nové realizaci. Požadavky na jednotlivé části modulu jsou znovu analyzovány a upraveny pro splnění nových potřeb. Výsledkem práce je funkční řešení řídícího modulu pro zařízení s OS Linux. Modul byl znovu vytvořen s použitím jazyka C++ s důrazem na jeho rozšiřitelnost a počítá s možným nasazením na dalších podobných zařízení.

Klíčová slova Linux, SNMP, NetSNMP, web, HTTP, libmicrohttpd, vestavěné zařízení, řídící modul, C, C++, SH4

Abstract

The work deals with remote control of devices with Linux OS. Gradually resolves possible implementations of remote control of network elements through a Web interface and SNMP. It closer focuses on libmicrohttpd and NetSNMP, with which help is solved technology. It is related to a device for creation of microwave connections, based on SH4 platform. Realization begins with environment, which was used for creation of the device firmware and gradually modifies and extends it. Original request of extending of the existing module with SNMP support and improved data collection gradually changes to complete reimplementation. Requirements on individual parts are again analyzed to meet new requirements. The result is a working solution of control module for devices with Linux OS. Module was recreated using C++ with emphasis on its extensibility and with possible deployment on other similar devices.

Keywords Linux, SNMP, NetSNMP, web, HTTP, libmicrohttpd, embedded device, control module, C, C++, SH4

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato práce se zabývá úpravami softwarového řídícího modulu společnosti SVM Microwaves s.r.o. . Modul je součástí firmwaru zařízení SkyWire pro tvorbu mikrovlnných spojů a poskytuje rozhraní pro jeho vzdálené ovládání.

Za dobu své existence však prošel značným vývojem a jeho stávající podoba začíná vykazovat problémy jak co do stability, tak i rozšiřitelnosti. Kromě potřeby vyřešit problémy s občasnými pády softwaru vznikly postupně také požadavky na úpravy a doplnění funkcí.

Cílem této práce je požadavky v co největší míře naplnit a vytvořit funkční řešení s přehlednějším kódem a lépe zpracovanými funkcemi. Modul by se měl dostat do použitelného stavu, ve kterém poskytne pojítku SkyWire možnosti pro vzdálenou správu a bude možné nad ním dále stavět.

Základem bude rozšířit modul o rozhraní SNMP V3. Podpora SNMP V1 je již přitom nějakým způsobem realizována, není ale jasné, zde bude možné V3 jednoduše přidat. V úvahu pak připadá úprava použitých knihoven, nahrazení jinými nebo implementace vlastního řešení.

Bude třeba také rozšířit řešení sledování informací ze zařízení. Stávající implementace k nim již dokáže přistupovat a umožňuje sledovat, zda jsou hodnoty v povolených rozsazích. To je třeba doplnit a zavést sledování vývoje hodnot v historii a umožnit i složitější kontroly. Upravené řešení by mělo umožnit například pracovat s průměry hodnot.

Modul již nyní poskytuje informace uživatelům prostřednictvím webového rozhraní. Jeho vývoj je ovšem aktuálně poměrně náročný a pomalý a bylo by dobré se nad ním znovu zamyslet a dokončit jeho realizaci. V aktuálním stavu jej bude třeba alespoň rozšířit o podporu dalších akcí pro doplnění možností konfigurace zařízení.

Samotná konfigurace by měla být přesunuta do flash paměti, pro niž byl v minulosti vytvořen novější ovladač [1]. Ten oproti původnímu efektivněji provádí čtení a zápisy, což by mělo umožnit konfiguraci po změnách bezpečně ukládat a obnovovat po restartu zařízení.

Došlo také k vývoji ohledně možnosti aktualizace firmwaru zařízení [2]. Vznikl návrh a částečná implementace, na kterých je nyní třeba dále zapracovat a v rámci možností je do modulu integrovat. Výsledné webové rozhraní by mělo umožnit práci s aktualizačními balíčky.

Proběhla také bezpečnostní analýza pro firmware i zařízení jako takové [3]. Z ní vzešla sada doporučení, která by měla být zohledněna a v rámci možností zapracována. Zároveň by měla být pro modul vytvořena další analýza, která původní doplní o nově vzniklá rizika.

Kapitola 1

Analýza

Začneme analýzou zadání a prostředí, ve kterém se budeme pohybovat. Vycházet budeme z informací získaných komunikací s vedoucím práce, kolegy, kteří se účastnili souvisejících prací a ze samostatného zkoumání původního modulu a sestavovacího prostředí firmwaru.

1.1 Zavedení pojmů

Pro pochopení práce bude třeba znát základní pojmy a principy z této oblasti. Některé z nich tu v krátkosti zavedeme.

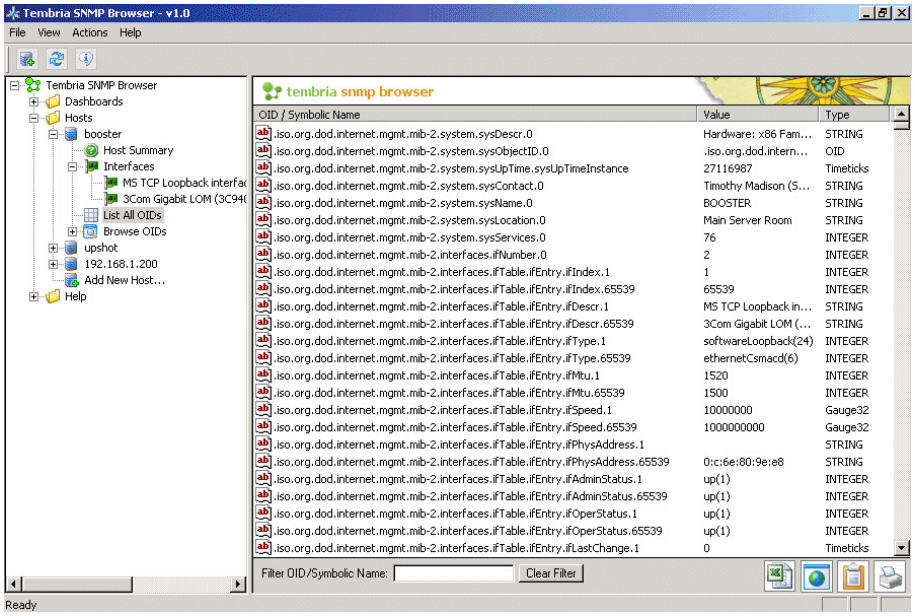
1.1.1 SNMP

Součástí práce je implementace rozhraní pro přístup pomocí protokolu SNMP [4]. Ten se v této oblasti velmi rozšířil a potkáme se s ním i u mnoha dalších zařízení.

Data jsou v něm dostupná přes jedinečné identifikátory, označované jako OID, které jsou tvořeny čísly, oddělenými tečkami. Podle implementace a přístupových práv je možné přes OID získávat ze zařízení informace a případně je i měnit.

Dohromady tvoří OID hierarchickou strukturu, jejíž jednotlivé části jsou popisovány MIB tabulkami [5]. Tam zároveň k jednotlivým záznamům nalezneme další informace a jejich textová označení. Bývají k dispozici jako textové soubory s obsahem v definovaném formátu.

Pro přístup k datům existují pak řádkové i grafické nástroje. Ukázka jednoho z nich je na obrázku 1.1.



Obrázek 1.1: Nástroj pro prohlížení SNMP hodnot. Zdroj: [6]

1.1.2 HTTP

V rámci řešení webového rozhraní pro nás bude důležitý protokol HTTP. Jde o textový protokol, používaný pro komunikaci s webovými servery[1].

Moderní webové aplikace pak často nepoužívají při přenosu informací jen klasické HTML, ale využívají jiné, snáze strojově zpracovatelné formáty. Zvlášť u webových aplikací se díky rozšíření jazyka JavaScript budeme setkávat s formátem JSON [7]. Pro práci s ním pak existuje mnoho knihoven, což značně ulehčuje jeho použití.

1.1.3 XML

Při práci se strukturovanými daty se často setkáme také s formátem XML [8]. Jde opět o velmi rozšířený formát pro uložení dat, který má zavedený způsob zapisování a navíc existují různé způsoby, jak je možné zapisovat kontrolní pravidla pro strukturu [9]. To se může hodit zvlášť ve firemním prostředí. Z hlediska způsobu čtení XML se pak používají prakticky dva možné přístupy [10].

Jeden je orientován spíš na sekvenční čtení, nesnaží se vytvářet v paměti objektový model a je vhodný při zpracování velký souborů. Implementace se pak označují jako SAX parser.

[1] HTTP: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Druhý naopak soubor načte do paměti a vytvoří z něj strukturu objektů. Přístup k ní je přitom pak často programově jednodušší, protože je bližší tomu, jak běžně pracujeme s objekty, není ale vhodný pro velké soubory. Implementace se označují jako DOM parser.

1.1.4 CGI

Při propojování webového serveru a programů, které umí generovat dynamické stránky, je možné využít CGI (Common Gateway Interface)[11]. Při požadavku na webový server je pak předán ke zpracování programu, který jej může zpracovat a vrátit odpověď v požadovaném formátu.

1.1.5 Readers Writers Problem

Při paralelním přístupu se často najdou místa, kde bychom mohli využít čtení dat z více vláken najednou a pak bychom mohli chtít povolit přístup více čtenářům nebo jednomu písaři. S tím je ale spojeno mnoho problémů. Roste množství potřebné paměti, nutnost zamykání a hrozí přitom, že se budou čtení nebo zápisy zpomalovat podle toho, čeho bude zrovna více.

Na tuto problematiku existují různá řešení. V literatuře pak bývá často označována jako Readers Writers Problem[12].

1.1.6 Buildroot

Bude se tu objevovat také pojem Buildroot[13]. Jde o systém, který zjednodušuje tvorbu distribucí OS Linux pro vestavěná zařízení, který umožňuje pomocí konfiguračních souborů vybrat potřebný software pro výslednou distribuci a způsob, jakým má být sestaven pro nějaký vybraný hardware. Mnoho nastavení lze přitom provádět pomocí dodaných grafických nástrojů [13].

Více se problematikou jeho použití zabývá práce o bezpečnosti pojítka [3].

1.2 Rozbor zadání

Po přiblížení pojmů se podíváme na jednotlivé body zadání, jejich význam a vyplývající úkoly.

1. **Analyzujte aktuální stav řídícího modulu pojítka SkyWire.**

   Pracujeme na řídícím modulu pro pojítko SkyWire. Jde o software, který se již v nějaké podobě používá a je jej třeba doplnit o další funkce. Z toho důvodu se nejprve seznámíme s jeho stavem, strukturou, stávajícími funkcemi a použitelností řešení.

2. **Na základě analýzy rozšiřte původní řešení o:**

   - **rozhraní SNMP V3,**
     Původní software již poskytoval rozhraní SNMP V1. Řešení ale nejspíš bohužel nebylo dostatečně testované a při dlouhodobém používání začalo vykazovat různé chyby a docházelo k výpadkům. Navíc neposkytuje podporu pro vyšší verze protokolu, které jsou ale přitom na jiných zařízeních běžně dostupné.
     Není ale zatím jasné, zda bude jednoduché dodání podpory vyšších verzí SNMP možné. V úvahu pak připadá nahrazení knihovny nebo implementace vlastního řešení. Vedoucím bylo doporučeno použití knihovny NetSNMP[14], která SNMP V3 podporuje, byla v době psaní práce aktivně vyvíjena a původní modul přitom již její starší verzi využíval.
   - **monitorování parametrů pojítka a zaznamenávání historie jejich hodnot,**
     Stávající řešení již implementuje sběr a kontrolu hodnot parametrů a ke každé načítané číselné hodnotě je možné doplnit minimum a maximum. Proti němu jsou pak data porovnávána a při problému je možné zasílat SNMP trap na zadané umístění.
     Toto řešení ale již není dostačující. Je třeba poskytnout také možnosti pro zápis složitějších kontrol, jako jsou hodnoty dlouhodobých průměrů, a vytvořit prostor pro další druhy následných akcí.
     Hodnoty z průběžného vývoje by pak bylo možné poskytovat i ve formě grafů přes webové rozhraní, stávající modul ale informace neukládá. Bude tedy třeba doplnit také ukládání agregovaných hodnot a možnost mít jejich vlastní specifické kontroly s podporou alespoň základních matematických výrazů. Ke každé pak musí být možné doplnit také sadu akcí při splnění podmínky, kde základní bude SNMP trap na zadané umístění.
   - **rozhraní pro komunikaci s webovým rozhraním,**
     V rámci původního řešení existují pro zpřístupnění webového rozhraní dva programy. Jeden řeší sběr dat ze zařízení a jejich lokální poskytování. Druhý je pak CGI skript, který data z prvního získává, generuje webové stránky v podobě HTML a poskytuje je klientům přes webový server. Jejich vzájemná komunikace je řešena pomocí Linuxových soketů [15].
     Implementace standardního webového rozhraní pomocí CGI skriptů ale zpomaluje vývoj. Je poměrně náročné jej začít významněji měnit a dále rozšiřovat. Místo toho by tedy bylo dobré upravit modul tak, aby bylo možné tvořit webové rozhraní samostatně a s hlavním procesem by pak komunikace probíhala pomocí nějakého jednodušího rozhraní, které nevyžaduje práci s daty na úrovni bajtů.

- **podporu pro aktualizaci zařízení.**

  Firmware zařízení je potřeba aktualizovat. Tento proces by bylo přitom vhodné mít možnost provádět i vzdáleně po síti, bez přímého přístupu k desce zařízení.

  Stávající řešení toto přitom neimplementuje. V rámci související práce [2] však již vznikl návrh, jak by toho bylo možné docílit. Tato práce by jej měla převzít a v rámci možností realizovat.

3. **Rozšíření implementujte a integrujte je do sestavovacího prostředí pojítka SkyWire.**

   Stávající řešení používá pro sestavování výsledného firmwaru prostředí Buildroot. To je třeba zachovat a i po provedení všech potřebných změn musí být možné firmware automaticky sestavovat s upraveným řídícím modulem.

4. **Zaměřte se na problematiku zabezpečení a analyzujte možné typy útoků na implementované řešení.**

   Výsledné řešení implementuje rozhraní, která budou dostupná po síti. Tím se případným útočníkům dostává dalších cest, kterými mohou ovlivnit funkčnost modulu a tedy i celého zařízení.

   Tyto cesty je třeba najít a analyzovat útoky a jejich rizika. Výsledná analýza doplní již existující práci k zabezpečení celého zařízení [3].

1.3 Předcházející práce

Samotná práce navazuje na předchozí tvorbu. Z ní přebírá principy, spojuje jejich výstupy a dotváří celistvý systém.

1.3.1 Sestavovací prostředí, programové vybavení a bezpečnost

Jedna z nich byla zaměřena na zabezpečení samotného pojítka [3]. Obsahuje jak důležitý popis jeho samotného hardwaru, tak i analýzu jeho zabezpečení. Tato práce by z ní měla přebrat principy a dodržet je pro zachování přijatelné míry bezpečnosti.

Kromě toho se zabývá také programovým vybavením. To je s touto prací velmi spjato, protože nejspíš bude třeba přidat další knihovny a dále upravovat nastavení existujícího prostředí. Po integraci upraveného modulu musí být přitom možné dál provádět automatizované sestavování celého firmwaru a jeho používání na cílovém hardwaru.

1.3.2 Flash

Jako persistentní úložiště slouží v námi používaném zařízení flash paměť. Práce s ní je ovšem velmi omezená kvůli její rychlosti a životnosti. Problematiku jejího používání částečně řeší práce [1]. Jejím výsledkem je nový ovladač, označovaný jako BFTL, který zpřístupňuje paměť po sektorech a výrazně tak urychluje zápis i prodlužuje životnost.

Jednotlivé alokované sektory jsou k dispozici jako zařízení v OS Linux a můžeme je tedy používat pro čtení i zápis. Musíme si ale sami řešit jejich obsah, identifikaci konce dat a hlídat jejich maximální velikost, protože pro sektor není použit žádný souborový systém.

Práce s pamětí není tedy ani tak úplně snadná a rychlost i životnost jsou stále nízké na časté zápisy. Budeme ji ale moc využít pro ukládání konfigurace a její obnovu po restartu zařízení.

1.3.3 Aktualizace firmwaru

Poslední práce byla zaměřena na možnosti vzdálené aktualizace firmwaru [2]. Jejím výstupem byl návrh řešení aktualizačních balíčků, jejich obsahu a zabezpečení. Výsledkem byla knihovna a navržená rozhraní, která je nutné dokončit a zapracovat do řídícího modulu.

V této části je přitom známé omezení, že samotný proces přepisu paměti bude muset proběhnout na jiné úrovni. Úkolem této práce je tedy pouze připravit most mezi uživatelem, přistupujícím přes webové rozhraní, a budoucím procesem, který zajistí zápis nového firmwaru na straně zařízení.

1.4 Řídící modul

Dále se blíže podíváme na řídící modul pojítka, jehož úprava je hlavním předmětem této práce.

1.4.1 Zdrojové kódy

Začneme zdrojovými kódy, které prošly dlouhým vývojem. První verze byla vytvořena okolo roku 2003 v jazyce C. Nemohlo tak být využito možností objektů a dědičnosti. Místo toho v něm nalezneme dlouhé větve podmínek s často špatně čitelným kódem a již nepoužívanými, a částečně zakomentovanými, bloky. Rozklíčování zamýšleného fungování kódu bývá někdy poměrně náročné.

Některé části byly postupně přepracovávány do objektů s použitím C++. Zdá se ale, že to ale vedlo spíš, než k zpřehlednění, k jeho dalšímu zkomplikování. Není jednoduché poznat, které části kódu se teď skutečně mají používat ani v jak funkčním stavu doopravdy jsou. Přepis bude představovat ještě hodně práce.

1.4.2 Hlavní proces

Původní modul přidává svoje funkce přímo do zdrojových kódů starší verze NetSNMP démona. K inicializaci dochází voláním vlastní spouštěcí funkce při jeho inicializaci, která zaregistruje obslužnou funkci pro odpovídání na dotazy do části tabulky SNMP a zároveň spustí oddělená vlákna pro sběr dat ze zařízení a obsluhu komunikace pro webové rozhraní. Tím inicializace v hlavním procesu končí a pokračují funkce NetSNMP.

1.4.3 Uložení v paměti

Informace modulu jsou pak přístupné přes globální datové struktury a přístup k nim je vždy obalen exkluzivním zámkem (mutex) [16]. Samotná data obsahují konfiguraci zařízení a nasbírané informace.

Tato varianta vyžaduje pozornost při práci, protože neopatrnou manipulací může snadno dojít k poškození obsahu. Výhodou ale je, že nevyžaduje další režii, která by jinak byla třeba například pro oddělení databáze do samostatného procesu a zavedení dalšího rozhraní.

1.4.4 Konfigurace

Důležitou částí modulu je konfigurace a její zpracování. Její obsah je uložen v textovém souboru, který je součástí firmwaru. Jeho obsah ovlivňuje hlavně sbíraná data a vzhled webového rozhraní, které je zde popsáno.

Pro uložení informací byl přitom využit přímo konfigurační soubor NetSNMP démona. To do značné míry určilo, jakou bude mít konfigurace strukturu. V souboru jsou dodefinovány pro řídící démon specifické identifikátory typů řádku. Na každém řádku jsou pak informace odděleny podobně, jako v běžném CSV souboru, kde jako oddělovač byl použit středník.

Zvolený formát je tak poměrně snadno strojově zpracovatelný. Není třeba řešit načítání složitějších struktur, jako například u XML, a pouze musíme mít dobře vydefinované obsahy sloupců.

Při kombinaci odlišných typů dat pro stejný typ řádku ale může být velmi nepřehledná a těžce udržitelná. Již ve stávající podobě obsahuje vzniklá tabulka mnoho prázdných polí. Další rozšiřování přitom může vyžadovat doplnění dalších sloupců, které úpravy dále zkomplikují.

1.4.5 Data ze zařízení

Konfigurace definuje datové položky, které lze ze zařízení získat. U každé je zadán typ, na který se váže použití informací ze sloupců v konfiguračním souboru, a informace ke zdroji dat.

Zdrojů dat je přitom více. Základním je I^2C sběrnice, kde se definuje adresa a počet bajtů. Je zde ale také například systémový čas. Informace jako verze firmwaru jsou pak také získávány přímo z konfiguračního souboru.

Problémem pak je, že se zde nachází i informace, jako adresa pro síťové rozhraní, kterou bychom mohli potřebovat změnit. Taková změna ale nebude trvalá, protože konfigurační soubor se obnoví po restartu zařízení.

Všechny položky jsou pak přístupné přes globálně dostupný seznam, pro jehož ochranu je připraven jeden výlučný zámek (mutex). Nad ním probíhá při požadavcích na čtení dat vždy opakované sekvenční vyhledávání.

Sběr dat pak probíhá v samostatném vlákně. Dochází v cyklu k opakovanému zamykání datových struktur přes společný zámek, aktualizaci dat všech položek z konfigurace, odemčení a uspání vlákna na zadaný čas.

1.4.6 Datové rozhraní

Pro zpřístupnění dat webovému rozhraní zavádí modul další vlákno. Poskytuje jak aktuální hodnoty jednotlivých datových položek, tak i různé doplňující informace o jejich struktuře, ze kterých pak CGI skript generuje webové stránky.

Požadavky na data mají definovanou strukturu na úrovni bajtů. Obsahují vždy číselný identifikátor příkazu a doplňující číselnou informaci. Pro každou operaci je pak pevná specifická struktura odpovědi.

S daty se přitom pracuje po jednotlivých bajtech a využívá se znalosti jejich struktury, výsledná komunikace je tak poměrně efektivní. Nemusejí se přenášet jinak potřebné režijní informace, jako jsou jinak v textově orientovaných protokolech různé identifikátory hodnot a jejich oddělovače.

1.4.7 Webové rozhraní

Samotnou funkčnost webového rozhraní zajišťuje webový server z balíku BusyBox [17]. Je nakonfigurován tak, aby požadavky na určité adresy předával ke zpracování CGI skriptu modulu. Tím je zde program v C, který se stará o zpracování požadavku, získání informací přes datové rozhraní a jejich převod do webové podoby pro koncového uživatele.

Pro zjednodušení jejich generování přitom zavádí pro některé stránky jednoduché šablony. V nich pak provádí náhrady za definované textové identifikátory, před jejich poskytnutí jako odpovědi uživateli.

Tvůrci webových stránek se běžně s nutností jejich psaní v C pravděpodobně často nesetkají. Tím se může jejich tvorba a vývoj bohužel výrazně zpomalit, i bez toho by ale museli zohlednit malé množství paměti zařízení a nemohli by využít rozsáhlejší knihovny, které by se jinak mohli hodit například při tvorbě webových aplikací v jazyce JavaScript, a vývoj by tedy ani tak nebyl snadný.

1.4.8 Aktualizace firmwaru

V rámci této práce bude třeba řešit také aktualizaci firmwaru a bude tedy důležité zjistit, v jakém je tedy její řešení aktuálně stavu.

Webové rozhraní umožňuje nahrávat soubory a přepisovat obsah stávajícího souborového systému, což bude možné využít pro nahrávání aktualizačního balíčku. Vlastní aktualizační proces podle práce [2] ale dále neřeší a zůstane tedy na této práci.

1.4.8.1 Licenční politika

Implementace bezpečné aktualizace podle návrhu je přitom důležitá kvůli snaze o změnu obchodního modelu společnosti SVM s.r.o. . Jejím cílem je snížit náklady, potřebné na vývoj zařízení, a místo mnoha různě výkonných zařízení mít pouze menší počet těch výkonných. Cena pak bude záviset na firmwaru, který adekvátně omezí funkčnost, a klientům umožněna jeho aktualizace [3].

Tento prostor pro ušetření ale s sebou přináší další bezpečnostní problémy. Firmware totiž tak nesmí být možné jednoduše upravit nebo použít na více zařízeních. Způsoby, jak tomuto alespoň částečně zamezit, se zabývá práce o zabezpečení [3] a jsou dále rozpracovány v rámci řešení aktualizací [2].

V rámci této práce se pro ně budeme snažit připravit vhodné prostředí a v co největší míře je implementovat.

1.4.9 Zařízení

Dále se podíváme blíže na samotný hardware, který máme k dispozici pro testování. Podoba jednoho z prototypů je na obrázku 1.2.

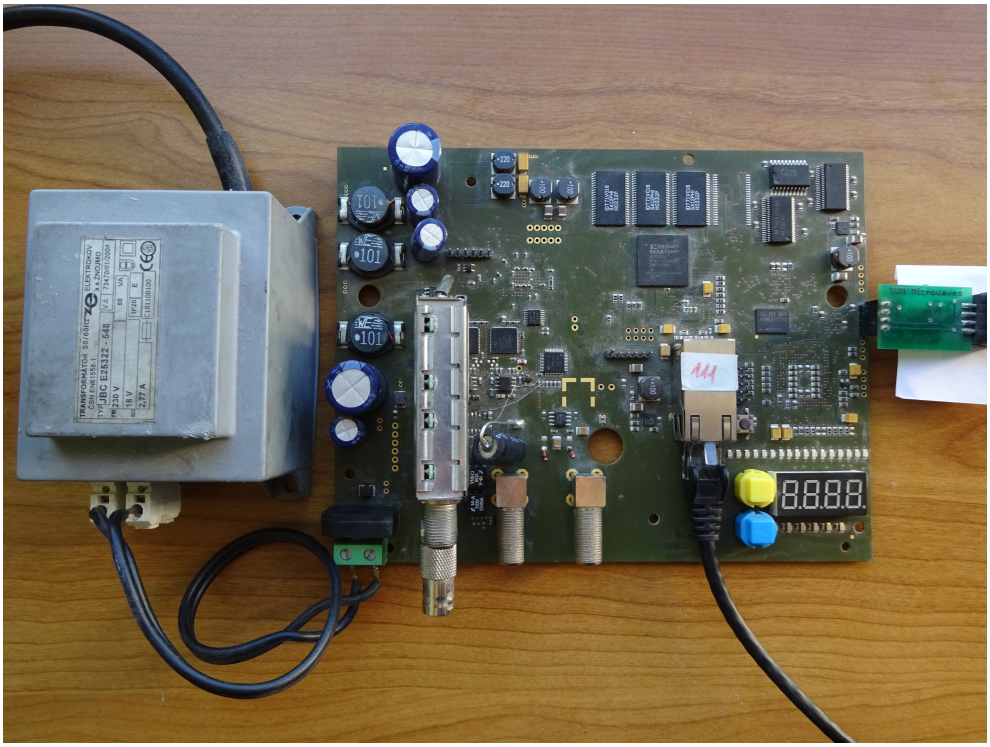
Jde o vysokorychlostní mikrovlnný datový spoj, označovaný jako pojítko SkyWire. Používá mikroprocesor SH7760 společnosti Renesas s architekturou SH4 (SUPER-H), který je taktován na 200 MHz[18]. K tomu je k dispozici 64 MB operační paměti a jako dlouhodobé úložiště pak slouží paměť flash s kapacitou 16 MB.

Zmíněné zdroje pojítka můžeme bez větších starostí využívat, slouží totiž pouze k jeho správě a o vlastní datové přenosy se starají samostatné FPGA jednotky. Komunikace s nimi probíhá přes I^2C sběrnici, která je tak pro nás klíčovou při přístupu k informacím a jejich nastavování. Podrobnější rozbor je součástí práce o bezpečnosti [3].

V rámci této práce je klíčové množství zdrojů a použitá architektura. Jejich kombinace a aktuální konfigurace firmwaru budou rozhodovat o tom, jaké funkce a knihovny bude možné při dalším vývoji modulu použít.

1.4.10 Firmware

V analýze budeme pokračovat firmwarem. První náhled získáme jednoduše přihlášením přímo ke konzoly pojítka a prozkoumáním souborového systému. Detaily k nastavení a verzím pak získáme přímo z konfiguračních souborů Buildroot a grafického rozhraní nástroje menuconfig.



Obrázek 1.2: Obdržená verze desky pojítka SkyWire. Zleva připojený zdroj, vlastní deska s konektorem pro Ethernet a vpravo výstup pro připojení na konzoly přes RS-232 do USB k počítači.

Z informací je vidět, že ze systému bylo odstraněno mnoho jinak standardních nástrojů i knihoven. Pro programy v jazyce C je k dispozici knihovna uClibc 0.9.31 se starší implementací vláken LinuxThreads [19].

1.4.11 Omezení

Díky použití systému OS Linux máme zjednodušený vývoj. Můžeme testovat na jiné platformě, jako je třeba x86, kde budeme mít také OS Linux, bez nutnosti opakovaně křížově kompilovat a spouštět kód vždy na zařízení. Je ale třeba brát ohled na výpočetní náročnost, nároky na paměť a zápisy do paměti flash.

Výkon použitého procesoru je ve srovnání s dnešními poměrně malý. Musíme tedy dobře zvážit, jaké činnosti budeme jak implementovat. Dobré ale je, že i když přetížíme procesor, výpočty budou pokračovat a „pouze“ si déle počkáme na výstupy. Riziko je tak prakticky pouze ve vypršení spojení a nižší kvalitě sbíraných dat kvůli větší různosti rozestupů mezi jejich sbíráními.

Podobně jsme na tom s pamětí, které je rovněž málo. Když nám ale v průběhu práce nebude stačit, systém začne ukončovat některé procesy. To

nám přitom může způsobit značné problémy a vést až k nutnosti restartovat zařízení. V průběhu implementace tedy budeme muset nároky zohlednit při volbě datových struktur a dostatek paměti hlídat a testovat.

Při uvažování funkcí musíme také zohlednit životnost a rychlost flash paměti. Problémy s ní sice řeší práce [1], i přes to je ale zápis velmi pomalý a způsobuje značné prodlevy při provádění různých akcí. Její opotřebování by pak znamenalo ztrátu schopnosti ukládat konfiguraci modulu a byla by nutná její výměna.

1.5 Požadavky

Ze zadání práce, průběhu analýzy a komunikace s vedoucím vyplynula sada požadavků, na které se budeme muset zaměřit. Doplníme navíc také požadavky, které se ukázaly jako zajímavé k řešení, ale ze zadání přímo nevyplývají.

Budeme je označovat notací: **[F/N]-[P/N]-X-00**. První část určí, zda jde o funkční či nefunkční požadavek. Druhá zda je povinný nebo není. Třetí kategorii, do které požadavek spadá, a čtvrtá pak požadavky v kategorii číselně odliší.

1.5.1 Funkční

Začneme funkčními požadavky. Těmi vydefinujeme, jaké funkce by měly být k dispozici po dokončení funkčního řešení a jak by se měly chovat.

1.5.1.1 Konfigurace

Základní kategorie se týká práce s konfigurací pro konkrétní zařízení.

F-P-K-01 Software bude nastavitelný prostřednictvím konfiguračního souboru, který bude součástí firmwaru.

K tomu by bylo ale dobré využít i nové možnosti zapisovat na flash. S tím přitom dosavadní řešení nemohlo příliš počítat, protože bez BFTL [1] nebyl zápis ze systému rychle a bez zbytečného přepisování obsahu možný.

Navíc by bylo dobré zvážit nějakou jinou, udržitelnější a rozšiřitelnější strukturu. Z konzultací vyplynula jako vhodná struktura XML, která by byla před uložením komprimována do formátu, který by podporoval z OS Linux nástroj gzip.

F-N-K-01 Modul umožní uložit měnitelnou část konfigurace trvale do flash paměti.

F-N-K-02 Konfiguraci v paměti flash bude možné měnit a znovu uložit.

F-N-K-03 Konfigurace bude mít strukturu XML.

F-N-K-04 Konfigurace bude komprimovaná do formátu, podporovaného nástrojem gzip.

1.5.1.2 Aktualizace

Funkční požadavky máme také na proces aktualizace firmwaru. Jsou směřovány hlavně na zachování myšlenek z návrhu.

F-P-A-01 Modul bude kontrolovat integritu aktualizačního balíčku.

F-P-A-02 Modul neumožní aktualizovat na firmware, který nebude správně zabezpečen.

1.5.1.3 HTTP

Další kategorie se týká webového rozhraní.

F-P-H-01 Software poskytne funkční webové rozhraní.

F-P-H-02 Umožní stahovat aktualizační balíček.

F-P-H-03 Umožní nahrávat balíček s firmwarem.

F-P-H-04 Umožní upravovat nastavení SNMP.

F-P-H-05 Číst data ze sledování.

F-P-H-06 Měnit nastavení Ethernetového rozhraní. (IPv4 adresa, síťová maska, brána, MAC adresa)

1.5.1.4 SNMP

Patří sem také požadavky, související s podporou SNMP.

F-P-S-01 Software poskytne možnost nastavit přístup přes rozhraní SNMP V3.

F-P-S-02 Přes SNMP budou přístupné průběžné (historické) hodnoty datových položek.

F-P-S-03 Modul implementuje možnost zaslat SNMP trap jako reakci na splnění podmínky ze sledování.

1.5.1.5 Data

Další požadavky se zaměřují na práci s daty a vynucují zachování stávajících možností a jejich další rozšiřování.

F-P-D-01 Modul bude nadále umožňovat správu většího množství různých datových položek.

F-P-D-02 Položky mohou být různých datových typů.

F-P-D-03 Položky musí být možné číst z různých datových zdrojů.

F-P-D-04 Položky musí podporovat čtení i zápis.

F-P-D-05 Datové typy i zdroje musí být možné kombinovat s ohledem na jejich význam u datové položky.

F-P-D-06 Datové typy musí podporovat konverze před zobrazením a před uložením do datového zdroje.

F-P-D-07 Modul bude schopen zaznamenávat historii vývoje hodnot položek podle nastavení.

F-P-D-08 Hodnoty historie bude možné číst prostřednictvím SNMP a HTTP rozhraní.

F-P-D-09 Modul umožní provádět akce podle výsledků základních matematických výrazů.
(), +, -, *, /, =, <, >, ==, <=, >=

F-P-D-10 Modul umožní v zápisech kontrol dat používat odkazy na hodnoty ze zaznamenané historie.

F-P-D-11 Modul implementuje jako akci ke kontrolám zaslání upozornění jako SNMP trap.

F-P-D-12 Sledování bude možné konfigurovat podle potřeb na daném zařízení.

Do této kategorie patří také požadavky, u nichž by bylo dobré zvážit alespoň návrh.

F-N-D-01 Systém bude podporovat přístup k diagnostickým datům ze síťového provozu.

1.5.2 Nefunkční

Vedle funkčních požadavků je třeba doplnit také další nefunkční.

1.5.2.1 Hardware

Základní kategorie je spojena s podporou hardwaru.

N-P-W-01 Software musí fungovat na dodaném hardwaru.

Obecně by ale bylo dobré, kdyby software byl co nejuniverzálnější.

N-N-W-01 Software nebude závislý na specifických vlastnostech použité architektury.

1.5.2.2 HTTP

K webovému rozhraní máme jeden spíš méně důležitý požadavek. Jeho realizace by byla zajímavá pro další vývoj, ale vyžadovala by přitom výraznější přepracování celého webového rozhraní.

N-N-H-01 K uživatelskému rozhraní přibude další s daty ve strojově snadno zpracovatelném formátu pro tvorbu moderních webových a dalších aplikací.

1.5.2.3 Aktualizace

Z hlediska aktualizací je důležitá bezpečnost a k ní se pak vztahují i požadavky.

N-P-A-01 Zabezpečení aktualizačních balíčků zachová principy z práce [2] .

1.5.2.4 Zdrojové kódy

Už v předchozích částech se objevovaly požadavky, které nejsou vyloženě nutné, ale bylo by vhodné je zapracovat pro další vývoj modulu. Nejnáročnější požadavky tohoto směru se pak vyskytují u zdrojových kódů. Bylo by totiž skutečně potřeba do nich významněji zasáhnout, aby se dohledali a odstranili důvody, které někdy vedou k pádům modulu, zvýšila se přehlednost a zjednodušila rozšiřitelnost celého řešení.

Navíc by bylo dobré nějakým způsobem oddělit implementaci řídícího modulu od knihovny NetSNMP. V tuto chvíli nelze při pohledu do složky zdrojových kódů říct, které patří modulu a které knihovně a není tedy ani jednoduše možné knihovnu nahradit novější verzí.

N-N-Z-01 Oddělit kódy NetSNMP a modulu tak, aby bylo možné knihovnu aktualizovat.

N-N-Z-02 Vyčistit a zdokumentovat zdrojové kódy.

1.6 Způsob implementace

Analýza a požadavky ukazují, že by bylo třeba udělat mnohem více, než jen upravit a rozšířit stávající kód. Modul je plánován pro další dlouhodobé užívání a rozvoj, aktuální verze přitom ale není plně funkční a už je těžce udržitelná.

Zdrojové kódy modulu byly přitom úzce spojeny s jednou ze starších verzí NetSNMP, která ale přitom prochází stále vývojem. Její aktualizace by přitom byla vhodná kvůli možným obsaženým bezpečnostním chybám i doplnění podpory SNMP V3.

Modul jako takový přitom může fungovat i samostatně a mohlo by jej být možné použít i na jiných zařízeních bez NetSNMP. Bude dobré se tedy podívat na možnosti, jak je NetSNMP možné využít dnes.

1.6.1 Přidání vlastního kódu do NetSNMP

Způsobů, jak přidat do NetSNMP vlastní kód, respektive požádat o možnost zpracovávat dotazy pro nějakou část hierarchie položek, je více. Jsou poměrně dobře popsány v dokumentaci [20]. Při psaní programu v jazyce C/C++ máme v podstatě dva možné přístupy.

1.6.1.1 Modul

Můžeme vytvářet samostatný modul, který se pak může stát staticky linkovanou součástí nebo dynamicky načítaným objektem. Tento přístup je velmi podobný tomu, který zde byl již použit, ale nevyžaduje tak výrazné zasahování přímo do kódu NetSNMP.

Jinak se varianty prakticky liší ve způsobu kompilování a spouštění. Statický modul se stane součástí jednoho výsledného binárního souboru. Dynamický je samostatný a načítá se při startu. Tento přístup nicméně není pro

náš případ příliš vhodný, protože bychom chtěli zachovat případnou možnost používat řídící modul bez NetSNMP.

1.6.1.2 Samostatný proces

Druhou variantou pak je mít samostatný proces a komunikovat s hlavním procesem SNMP prostřednictvím protokolu AgentX.

To tedy řeší problém implementace jako modulu. Máme samostatný program, který můžeme kompilovat, spouštět a ladit podle našich potřeb. Můžeme přitom výsledné řešení upravit i tak, že jej bude možné nasadit jinde, bez přítomnosti knihovny NetSNMP.

Nevýhodou je ale nutnost komunikace procesů, která bude představovat režii při obsluze požadavků.

1.6.1.3 Výběr

Je třeba se rozhodnout pro jednu z cest. Předběžně se zdá být varianta se samostatným procesem lepší. Umožňovala by větší volnost při implementaci i používání. Protokolem se prakticky nemusíme zabývat, protože je realizován knihovními funkcemi.

Problém by ale mohla být již zmíněná výkonnost kvůli přidanému rozhraní. Komunikace s vedoucím ale ukázala, že výkonnost rozhraní není kritickým parametrem. Důležité je, aby požadavky byly vyřizovány správně a bez výskytu vypršení spojení.

Testovací implementace obou variant ukázaly, že pokles výkonu není natolik velký, aby výrazně omezil použití tohoto řešení. Přínos, v podobě možné nezávislosti na NetSNMP, je ale přitom zajímavý.

Vydáme se tedy cestou samostatného procesu. Výsledné řešení pak v rámci testování změříme a v případě, že by byla rychlost příliš malá, bude možná nutné jej převést na jinou variantu. Protože prakticky se ale v podstatě mění pouze způsob zapnutí funkcí řídícího modulu a jeho ukončování, tak by to teoreticky mohlo být možné.

Jediné, na co je pak u varianty modulu třeba myslet, je automatické restartování snmpd při případném ukončení nebo dokonce pádu procesu. Modul by měl být schopen se i při případných problémech, které by mohly vést k jeho pádu, znovu spustit a vykonávat své funkce. Bude tedy třeba restartovat buď samostatný proces nebo hlídat přímo hlavní proces NetSNMP.

1.6.2 Bezpečnost a procesy

Při srovnání požadavků na bezpečnost a stávající implementace objevíme řadu rozdílů. Stávající modul využívá vláken, která sdílí paměťový prostor a bezpečný přístup do něj řeší zamykáním. Máme zde tedy poměrně malé množství režie, ale jdeme zároveň také proti požadavkům na bezpečnost, kde bychom

naopak měli mít vše oddělené a v ideálním případě v různých procesech s různými uživateli a spojené pouze lokálně rozhraními. Tak by se minimalizovali i škody při případném průniku útočníkem.

Výpadek vlákna navíc vede k pádu celého procesu. O jeho nové spuštění se sice může postarat skript, i tak ale přijdeme o již nasbíraná data a vše je nutné znovu inicializovat. U stávající implementace to ještě není takový problém, upravená ale ztratí obsah historie a to už by nám mohlo vadit.

Přidaná rozhraní a komunikace ale s sebou přináší také režii. Prakticky bychom museli implementovat lokálně přístupnou databázi v paměti a procesy, které by z ní četly a zapisovaly. Přitom ale stále pracujeme na zařízení s pouze jedním procesorem a jádrem. Budeme tedy muset najít nějaký kompromis.

Pokud pomineme riziko v podobě narušení systému vlastní chybou, na což budeme muset modul testovat, nejvyšší riziko bude na vnějších rozhraních. Při snaze o zvýšení bezpečnosti tedy budeme určitě muset oddělit procesy, komunikující s vnějším světem, a ostatní. V případě SNMP jsme již přitom použití samostatného procesu měli už v plánu a u webového rozhraní tomu tak již je.

Na vnitřní straně pak nároky na oddělení logických procesů snížíme. Pokud by útočník skutečně měl schopnosti a prostředky, pak by se postupně dostal přes jakékoliv množství rozhraní. Některé procesy přitom budou muset běžet s vysokými právy, protože bude potřeba například nastavovat Ethernetové rozhraní nebo číst ze sběrnice. Použití superuživatele se tedy nevyhneme a tyto části tedy ponecháme v prostoru jednoho procesu pod více vlákny.

1.6.3 Zapracování změn

Zde zmíněné návrhy a rozhodnutí se projeví na architektuře konečného řešení. Implementace změn bude vyžadovat rozsáhlé úpravy kódu, který ale přitom je již poměrně starý a není příliš dokumentovaný.

Bez testování nelze jednoduše říct, které části se používají a které již ne. Když navíc zohledníme, že v minulosti již probíhaly snahy o přepsání s použitím objektů, možná by stálo za zvážení, zda se na to nezaměřit více a neimplementovat modul zcela znovu a původní použít pouze pro inspiraci.

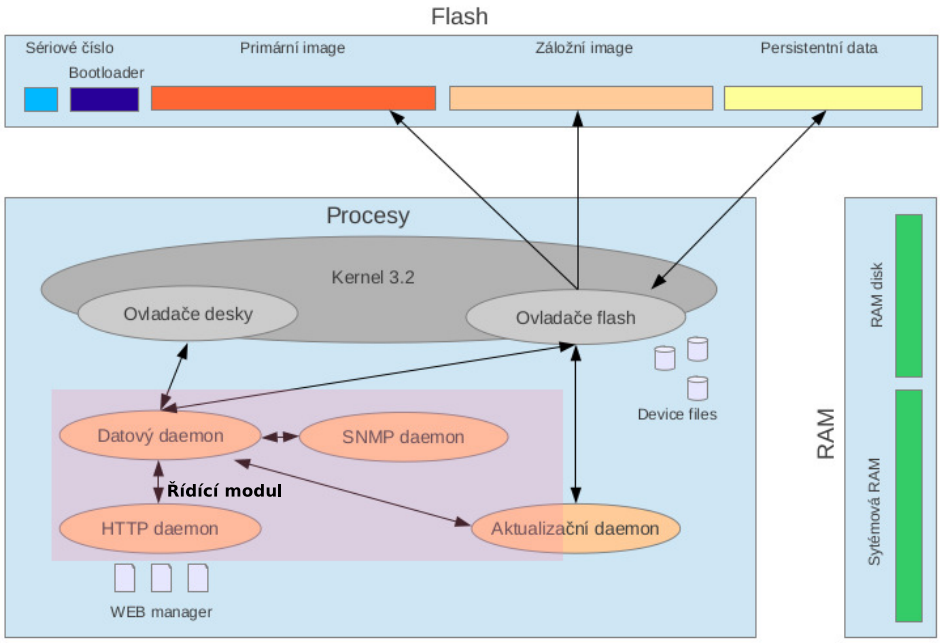
Výhodou postupného přepracovávání by bylo, že bychom měli stále relativně funkční řešení. Změny bychom mohli zapracovávat postupně bez seznamování se všemi částmi modulu. V původním stavu toho ale nezůstane mnoho. Postupně by bylo dobré zcela změnit přístup k aktuálně globálním strukturám a nakonec by nám nejspíš zůstaly pouze části na nejnižší úrovni, jako je práce se sběrnicí.

Nová implementace by vyžadovala tvorbu nového návrhu i u základních částí, které již nějakým způsobem fungují. Při vývoji by se ale dal snáze odstínit vliv již existujícího kódu a jeho podoby a to by mohlo usnadnit tvorbu udržitelnější podoby.

Po dalších konzultacích s vedoucím práce bylo rozhodnuto, že další implementace bude provedena zcela od začátku. Nový kód z původního přejme principy, postupy řešení některých problémů a zkombinuje je s výstupy z předcházejících prací.

Cílem tedy již nebude zachovat vnitřní rozhraní mezi částmi řešení a provést pouze úpravy a rozšíření, ale realizovat celý modul, který umožní lépe provádět vzdálenou správu po požadovaných rozhraních a bude snáze rozšiřitelný.

Graficky by se vliv práce tedy dal shrnout upraveným původním návrhem struktury 1.3. Výsledek bude nahrazovat původního datového, SNMP a HTTP démona, doplní propojení na aktualizační proces a připraví půdu pro novou webovou aplikaci.



Obrázek 1.3: Zaznačen vliv práce na celkovou podobu. Původní zdroj: [3]

Kapitola 2

Návrh

V této kapitole přejdeme k návrhu realizace. Postupně se zaměříme na jednotlivé prvky, nutné pro následnou realizaci, a připravíme si řešení pro požadavky z analýzy. Zaměříme se přitom na modul ze všech pohledů, protože přecházíme z úpravy na jeho novou realizaci.

2.1 Zdrojové kódy

Začneme plněním požadavků na zdrojové kódy. Zde máme pouze N-N-Z-01, který splníme jednoduše tak, že se budeme k NetSNMP části chovat jako samostatnému sub modulu našeho řídícího modulu a necháme jej pracovat odděleně.

Jinak jsou základní požadavky na kód docela obecné. Při návrhu je potřeba pouze myslet na přehlednost a rozšiřitelnost, a zdrojové kódy i další materiály tedy budeme udržovat rozdělené do celků podle použití.

Abychom mohli vůbec začít tvořit, bude navíc třeba nějaký název pro náš modul. Označíme jej jako StationManager (na způsob správce zařízení) a v kódu oddělíme jeho části do, z označení odvozeného, jmenného prostoru „sm“.

2.1.1 Základní struktura

Základní strukturu souborů založíme na informacích z požadavků, které již známe. Víme, že budeme muset mít konfigurační soubory, zdrojové kódy a také nějaké testy. K tomu přibudou knihovny a pravděpodobně také různé další skripty. Samotné zdrojové kódy částí pak bude vhodné dále strukturovat pro přehlednost.

Prakticky přitom budou po kompilaci vznikat dva programy. Jedním bude řídící modul (mod), druhým pak obslužný program pro zpracování požadavků na webové rozhraní (http).

- buildroot - balíček pro Buildroot

  Do této složky umístíme balíček pro Buildroot. Bude udržovat informace o závislostech modulu a umožní jeho snadné začlenění do sestavovacího procesu.

- conf - konfigurace a související soubory

  Sem umístíme hlavní konfigurační soubor. Bude zde v čitelné XML verzi s komentáři a ve zkomprimované, kterou bude modul využívat při své práci.

- http - webový server

  Zde budou zdrojové kódy programu pro obsluhu požadavků webového rozhraní.

- lib - nezávislé externí knihovny

  Tady budou staticky linkované knihovny, které bude modul používat.

- mod - moduly systému

  Zde se budou nacházet samotné zdrojové kódy jednotlivých částí řídícího modulu.

- tests - testy

  Sem umístíme skripty, kterými bude možné provádět testování implementace.

- scripts - skripty

  V této složce budou případné pomocné skripty. Půjde o řešení sestavení konfigurace pro běh softwaru a startovací skripty pro OS Linux.

- www - webové stránky

  Sem přijdou soubory webového rozhraní. Půjde o různé HTML, JS a CSS soubory a případně obrázky.

V hlavním adresáři pak navíc bude hlavní vstupní soubor řídícího modulu, obsluhy pro web a sestavovací skripty. Pro sestavování využijeme autotools [21], které jsou podporovány použitým systémem Buildroot a pomohou nám implementovat, pro OS Linux standardní, sestavovací kombinaci konfigurace (./configure), sestavení (make) a instalace (make install).

2.1.2 Přístupy k informacím

Řídící modul bude dobré dále členit podle jednotlivých činností. Výsledkem budou různé obslužné objekty jak pro jednotlivé hardwarové prvky, tak ale i nasbíraná data nebo kontrolní procesy.

Abychom si zajistili dostupnost těchto objektů, zavedeme hlavní třídu, přes kterou se k instancím těchto objektů budeme moci v programu dostat. Budeme ji označovat jako „Machine“ a kromě přechovávání objektů ji použijeme také pro počáteční inicializaci celého řídícího modulu a zavedeme v ní funkce pro start i ukončení jeho funkce. Tak umožníme provést korektní ukončení činností modulu.

Při takovémto sdílení se často budeme muset potýkat se synchronizačními problémy. Při návrhu budeme muset na jejich korektní řešení klást velký důraz.

2.2 Konfigurace

Základním prvkem pro úspěšnou inicializaci a zpřístupnění funkcí modulu bude načtení konfigurace. Implementaci bychom ale přitom chtěli vylepšit a místo načítání z konfiguračního souboru SNMP už by měla být ve vlastním komprimovaném souboru, který si modul dokáže rozbalit. Původní formát na způsob CSV by měl být navíc nahrazen standardnějším XML (F-N-K-03).

Její načítání tedy upravíme. Umístění tohoto konfiguračního souboru budeme předávat modulu při spouštění jako parametr, samotnou dekompresi i převod XML do přístupnějšího formátu pak zajistíme použitím knihoven pro C++.

2.2.1 Práce s obsahem

Začneme kompresí dat XML podle F-N-K-04. Potřebujeme tedy komprimovat formátem, který podporuje nástroj gzip. Knihoven bychom jistě našli více, v systému ale již máme zlib a využijeme ji.

Dále by se nám hodila knihovna pro zpracování dat v XML formátu. Potřebujeme přitom nějakou pod licencí, která umožní i případné komerční využití, podporuje čtení i úpravy XML, poskytne jednoduché rozhraní a umožní obsah objektů vyexportovat a uložit opět do souboru v textové podobě. Mělo by tedy nejspíš jít o nějaký nenáročný DOM parser.

Takových knihoven je opět k dispozici mnoho. Nakonec byla vybrána PugiXML [22]. Při základních experimentech prokázala velmi použitelné rozhraní a velikost knihovny ani množství zabrané paměti po načtení XML přitom nepředstavoval problém.

2.2.2 Rozdělení konfigurace

Velká část konfigurace bude závislá čistě na aktuální verzi firmwaru a konkrétním zařízení, některé části ale bude třeba měnit (F-N-K-*). Konfiguraci tedy rozdělíme do dvou bloků.

Budeme mít jeden hlavní soubor. V něm budou všechny základní informace, včetně výchozího nastavení pro měnitelnou (uživatelskou) část, a bude součástí firmwaru.

Vedle něj pak budeme udržovat uživatelskou část, kterou umístíme do flash paměti. Při prvním startu po nahrání firmwaru do něj nahrajeme výchozí podobu a s jeho aktualizacemi budeme obsah přepisovat.

Pro správný přepis zavedeme parametr verze uživatelské konfigurace. Když se verze změní, obsah sektoru se přepíše a následně bude tedy zároveň nutné modul znovu nastavit.

2.2.3 Práce s flash pamětí

Pro práci s daty na flash paměti použijeme ovladač BFTL. Stejně jako u každé paměti, bude třeba i zde zvážit velikost prostoru pro ukládání a řešení situací, kdy by jí již nebyl dostatek.

BFTL přitom nezavádí souborový systém, ale pouze umožňuje alokovat prostor po sektorech fixní velikosti. Jako velikost sektoru bylo zvoleno 128 KiB a jeden takový byl pro uložení konfigurace vyhrazen. S konfigurací tedy budeme muset pracovat tak, aby ji do něj bylo možné bezpečně ukládat a správně načíst po restartu modulu, respektive celého zařízení.

Základem bude rozpoznání datového obsahu v sektoru pro načtení. K označení konce můžeme využít různých přístupů s použitím definovaných značek. Vzhledem k celkové velikosti si zde ale vystačíme i se zápisem počtu významných bajtů na začátek našeho sektoru a vyhneme se vyhledávání.

Zapisované XML může být poměrně velké. Bude se v něm totiž nacházet celá specifická část konfigurace, včetně konfiguračního souboru pro proces SNMP. Nezkušený uživatel, nebo útočník, by tedy mohl kapacitu sektoru chybou snadno vyčerpat, i když ji budeme komprimovat (F-N-K-04).

Abychom tomu alespoň částečně zamezili, budeme zavádět na úrovni jednotlivých funkcionalit limity na velikosti polí tak, aby šance byla minimální. V praxi ale může být potřeba někde hranice později posunout a tomuto riziku se stejně nevyhneme. Pro zajištění bezpečnosti tedy budeme před zápisem výslednou velikost dat muset ještě kontrolovat a v případě, že by se již do sektoru nevešla, ji neuložíme a zápis ukončíme s chybou.

2.2.4 Paralelní přístup

Nakonec nám již zbývá pouze problém s paralelním přístupem k datům konfigurace. Použitá knihovna vytváří v paměti objektovou reprezentaci, do které si následně na různých místech předáváme odkazy.

To může být problém, pokud s ní budeme manipulovat. Při práci pak mohou vznikat a zanikat objekty, které může ještě někdo jiný používat. Implementujeme zde tedy zamykání na způsob Readers-Writers Problem s preferencí písařů.

V kódu si pak musíme dát pozor, abychom si někde odkazy do objektové reprezentace nenechávali. Místo toho z ní budeme muset informace vždy buď přečíst a vykopírovat nebo zamykat a znovu hledat patřičný uzel s daty, abychom neriskovali chybný přístup do paměti.

2.3 Zařízení

Dále budeme muset nějak přistupovat k hardwarovým částem samotného zařízení, kterým budeme potřebovat zadávat různé požadavky.

Pro přístup k nim ale přitom může být třeba je nějak inicializovat a často také zajistit výlučný přístup, abychom je nedostali do nekonzistentního stavu. Příkladem může být I^2C sběrnice, ze které budeme hlavně sbírat data, nebo síťová rozhraní, u nichž bude třeba nastavovat síťové adresy.

2.3.1 Zpřístupnění v modulu

Pro každé zařízení bude třeba ukládat počáteční konfiguraci. Tu můžeme umístit přímo do hlavního konfiguračního souboru a její měnitelné části pak do uživatelské konfigurace. Každé zařízení bude muset mít minimálně číselný identifikátor pro další sdílení napříč řídícím modulem.

Objekty zařízení pak bude třeba vytvářet a inicializovat. K tomu si vytvoříme jeden správcovský objekt a v něm zaregistrujeme statické vytvářecí metody objektů jednotlivých zařízení pod textové identifikátory.

Tak budeme moci v konfiguraci zavést struktury pro jednotlivá zařízení, která bude možné přímo předat vnitřně správné metodě k vytvoření objektu zařízení pouze podle textových identifikátorů. Tam zároveň můžeme umístit případné kontroly, zda je konfigurace správná. Takto vytvořené objekty pak budeme udržovat přístupné přes objekt „Machine“ a jejich číselný identifikátor.

Pro zjednodušení přístupu k různým druhům zařízení, a možnosti jejich obměn při použití řídícího modulu jako takového někde jinde, zavedeme jejich základní hierarchii. Tak nemusíme limitovat například načítání konfigurace na BFTL sektor, ale bude stačit říct, že zdrojem musí být úložiště, ze kterého lze přečíst data. Kromě přístupu k souborům, našem případě konkrétně k flash paměti, budeme potřebovat i další zařízení.

Další kategorií budou síťová rozhraní. Na desce je k dispozici Ethernetový port, u kterého potřebujeme potřebovat alespoň adresu IPv4, bránu, MAC adresu a síťovou masku. I pro ně vytvoříme tedy samostatnou kategorii.

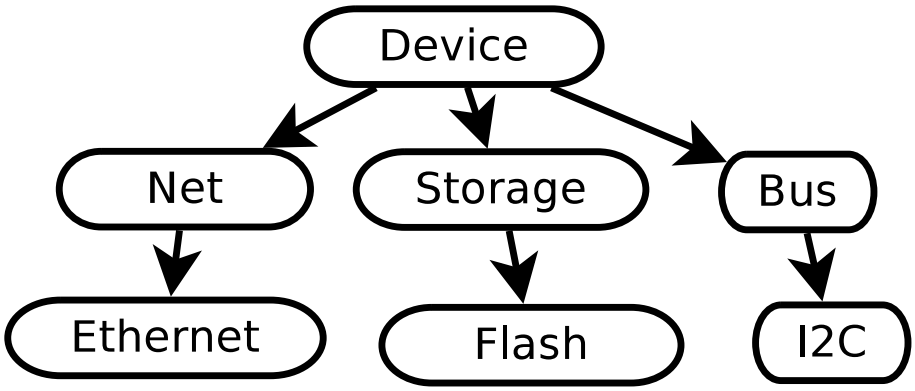
Z pohledu vlastního provádění změn nastavení pak budeme mít prakticky dvě možnosti, jak je implementovat. Mnoho nastavení lze provádět přímo z

C++[23]. Tento velmi efektivní přístup však není z hlediska kódu příliš průhledný a budeme prakticky znovu implementovat již existující nástroje, jako je ifconfig [24].

Pro zjednodušení tvorby a vyvarování se chyb je budeme spouštět přímo. Bude se sice muset spouštět externí příkaz, což bude pomalejší a vyžadovat další paměť, nepředpokládáme ale, že to bude nějak častá činnost a nemělo by to představovat problém.

Budeme muset také často přicházet do kontaktu se sběrnicemi, zde konkrétně I^2C. Zavedeme tedy pro ně obecně další kategorii a rozhraní jí vytvoříme podle námi použité. Budeme tedy prakticky potřebovat funkce pro čtení a zápis dat nějaké délky vzhledem k nějaké adrese.

Výsledný základ hierarchie znázorňuje obrázek 2.1.



Obrázek 2.1: Základní struktura zařízení modulu

2.3.2 Paralelní přístup

I u zařízení budeme muset řešit paralelní přístup. Způsobů by se dalo určitě nalézt více a i zde by bylo možné řešit například povolení více čtenářů. U všech tří zařízení se ale zdá být jistější mít přístup zcela výlučný. Zařízení tedy budeme zamykat standardně výlučně a v případě potřeby je možné to pro některá implementovat jinak. U nich pak bude vhodné jim vytvořit vlastní skupinu v hierarchii.

2.4 Data

Dále se zaměříme na práci se sbíranými daty ze zařízení. Zde z požadavků vyplývá nutnost situaci dramaticky zlepšit skrytím výpočetní logiky do objektů, zavedením dalších struktur pro držení historie a vylepšení implementace sledování.

2.4.1 Položky

Jednotlivé záznamy v naší databázi sbíraných dat budeme označovat jako položky. Každá bude mít svůj typ a zdroj. Vzájemně půjdou prakticky libovolně míchat, protože bez dalších informací budeme pracovat na úrovni bajtů.

2.4.1.1 Datové zdroje

Datových zdrojů budeme potřebovat více. Podle položek z původní konfigurace byly identifikovány následující.

- **Flash (Mem)**

  Základním zdrojem bude odkaz do uživatelské konfigurace. Půjde tedy prakticky o persistentní proměnnou, kterou bude možné přepsat při aktualizaci firmwaru nebo podle nastavení položek i uživatelem. Můžeme zavést například textovou položku s informací o fyzickém umístění zařízení.

- **Sběrnice (Bus)**

  Důležitým zdrojem bude I^2C sběrnice. Přes ni budeme schopni jak sledovat hardware desky, tak i provádět případnou konfiguraci datových přenosů.

- **Výpočet (Math)**

  Původní konfigurační soubor zaváděl také výpočetní typy. Zaváděl pak položky, jejichž hodnoty vznikaly například násobením hodnot z několika jiných. K tomu implementujeme samostatný zdroj, kterému bude možné zadat předpis s odkazy na další položky a bude schopen provádět jednoduché výpočty (F-P-D-09).

  Hodila by se nám k tomu nějaká otevřená knihovna. Po hledání byla vybrána [25]. Do ní bude stačit doplnit operátory porovnání a obalující objekt, který zajistí překlady proměnných a doplnění hodnot za proměnné ve výrazu z databáze před jeho vyhodnocením (více viz 2.4.8).

- **Net**

  Dodané zařízení je určeno k udržování síťového spojení mezi dvěma místy. Toho by bylo možné využít k průběžné výměně různých diagnostických dat (F-N-D-01), která budou mít většinou stejný nebo hodně podobný obsah. Při přístupu k informacím na jednom zařízení bychom tak měli hned k dispozici i ty z druhého konce. To může být zajímavé nejen proto, že se pak není nutné připojovat k zařízení na druhém konci, ale také proto, že by to mohlo umožnit distribuci informací, které by se tak neztratili ani při výpadku jednoho z nich, protože po zapnutí by se

k němu informace mohla znovu dostat. Z pohledu pojítka by se to dalo využít k šíření informace o porušení licenční politiky [3].

Z hlediska implementace by opět bylo možné využít I^2C sběrnice. Do hlaviček paketů komunikace by se přidávalo několik bajtů s tímto druhem informací z konkrétního rozsahu adres na sběrnici, na druhé straně by se pak průběžně promítaly opět do pevně daného rozsahu adres. Zápis i vyčítání dat by si zpracovával řídící modul a podle konečného množství informací, které by se takto musely přenášet, by se dalo uvažovat, zda stačí pouze příznaky nebo je třeba implementovat nějaký složitější protokol.

Při větším množství stahovaných dat by bylo možné toto řešení zkombinovat s využitím některého z dalších rozhraní. Zařízení by se vzájemně mohla dohadovat například přes webové rozhraní a data si poskytovat průběžně. Jejich stahování by pak mohlo probíhat periodicky podobně, jako je tomu při sběru dat lokálně. V případě, že by šlo pouze o výjimečné události, by se v komunikaci mohl nastavovat příznak, který by protistranu informoval, že si diagnostická data má stáhnout.

V tuto chvíli ale dál nepůjdeme. Původní řešení toto neimplementuje a realizace by přitom vyžadovat další komunikaci s vedením firmy a pravděpodobně i zásah do implementace vlastní síťové komunikace.

2.4.1.2 Datové typy

Datových typů zavedeme také více. Po seznámení s původní konfigurací byly identifikovány následující.

- Float - čtyřbajtové číslo v pohyblivé řádové čárce
- Double - osmibajtové číslo v pohyblivé řádové čárce
- Integer - čtyřbajtové celé číslo
- String - text proměnné délky

U některých položek mohou být později třeba speciální typy. Některé informace mohou totiž být po vyčtení například přes sběrnici v nějaké upravené podobě a pro jejich zobrazení jako čísla může být třeba je nějak přepočítat (například posunout hodnotu o nějaký počet bitů). Zavedeme tedy u typů konverzní funkce z i do formátu pro uložení. Ty pak budeme v patřičných situacích volat a tuto převodní vrstvu tedy zcela skryjeme v datovém typu.

Některé akce pak budou potřebovat další operace, které umožní skrytí složitosti, která je spojená s nutností různé implementace u různých datových typů. Příkladem může být uložení hodnoty z textové reprezentace. U typu String můžeme bajty uložit přímo, zatímco u typu Integer musíme nejprve

provést převod do číselné podoby a následně ještě provést konverzi na formát pro zdroj.

Konkrétní funkce, které bude třeba implementovat, se postupně ukáží při návrhu a realizaci. Datové typy jsou na jejich doplnění připraveny.

2.4.2 Historie

Po zavedení datových položek potřebujeme někam také uložit průběžnou historii jejich hodnot. K tomu využijeme objekty datových typů, do nichž přidáme tabulku historických hodnot podle konkrétního nastavení jejích rozměrů v hlavní konfiguraci, a budeme v ní udržovat přesné hodnoty na prvním řádku a víc a víc zprůměrované na dalších. Při zaplnění řádku vždy spočítáme průměr, hodnotu zařadíme do řádku vyššího a postup opakujeme.

Konečný možný obsah ilustruje tabulka 2.1. Zde máme celkem čtyři úrovně, kde na každém řádku je šedesát hodnot. Do prvního ukládáme nasbíraná data přibližně jednou za sekundu, do druhého pak každou minutu vložíme průměr z hodnot za tu poslední.

Úroveň	Rozestup	Délka úrovně
0	1 sekunda	1 minuta
1	1 minuta	1 hodina
2	1 hodina	2,5 dne
3	2,5 dne	150 dní

Tabulka 2.1: Chování historie hodnot dat

Výsledná délka úrovně však může být zavádějící. Při sběru dat provádíme pouze jednoduché uspávání, které ale není přesné a skutečná délka se může měnit podle aktuální zátěže. Při kontrole vývoje v historii je s tím třeba počítat a hodnoty brát s rezervou.

2.4.3 Organizace položek

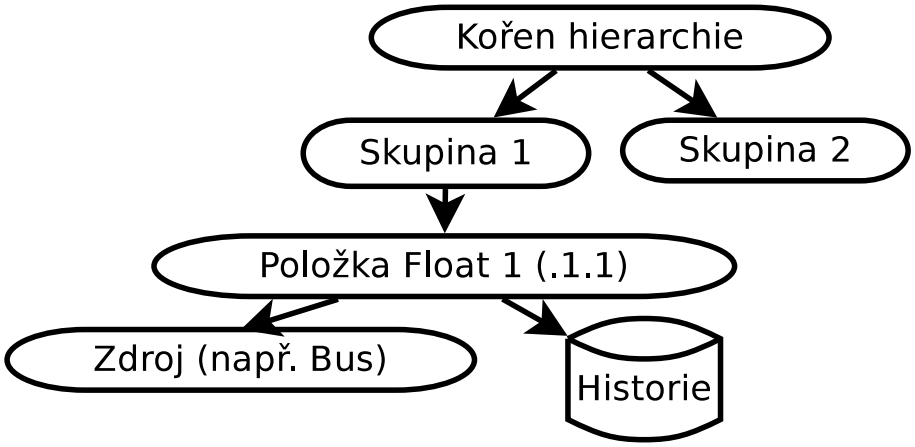
Data ve vnitřní databázi by mohlo být vhodné zpřístupňovat v nějaké strukturovanější podobě. Budeme-li poskytovat data z obou zařízení pro jeden spoj (lokálního i vzdáleného), mohlo by být dobré je odlišit i při poskytování přes SNMP.

Na úrovni konfigurace tedy umožníme vytvářet skupiny položek, které do sebe bude možné zanořovat a dohromady vytvoří hierarchii. K takto strukturovaným datům umožníme přistupovat i přes webové rozhraní skupinově a na jeden dotaz tak půjde vytáhnout z databáze i jejich větší množství s nižší režií.

Do těchto skupin se budeme potřebovat nějak odkazovat, aby bylo možné si o jejich obsah vzdáleně požádat. Pro SNMP budeme muset implementovat

dotazování přes OID a stejné značení pro jednoduchost použijeme i u webového rozhraní.

Výsledné položky tak budou vždy složeny z kombinace typu a zdroje. Ty budou organizovány do hierarchické struktury, kde každá úroveň bude označena číselným identifikátorem. Graficky je to znázorněno na obrázku 2.2



Obrázek 2.2: Sbíraná data a jednotlivé položky

2.4.4 Konfigurace

Hierarchie položek bude rovněž vyžadovat základní nastavení. Do hlavního konfiguračního souboru bude třeba doplnit jak základní informace o položkách pro poskytování webovému rozhraní, tak i například vazbu položek na informace a zdroje a jejich rozmístění v jejich struktuře.

Takto budeme moci mít jednu položku na více místech v celkové struktuře a bude možné mít jinak strukturované skupiny dat pro webové rozhraní a SNMP. Skupiny pro SNMP a web bude v konfiguraci stačit oddělit parametrem.

2.4.5 Zavedení položek

Po rozvržení položek i konfigurace bude třeba zařídit vytvoření patřičných objektů v kódu. K tomu využijeme informace o typech i zdrojích v hlavním konfiguračním souboru.

Připravíme si objekty s vazbou vytvářecích statických funkcí na textové identifikátory zdrojů a typů. Při přípravě položek pak budeme postupně iterovat přes konfiguraci, vytvářet položky, přiřazovat jim zdroje a zařazovat je do vnitřní databáze.

Vytvářecím funkcím můžeme přitom předávat přímo odkazy do konfigurace. Na jejich úrovni pak budeme provádět i případné kontroly a chybně zapsané položky můžeme přeskakovat.

2.4.6 Aktualizace informací

Informace v naší databázi budeme muset pravidelně aktualizovat. Využijeme přitom toho, že naše hlavní vlákno nebude mít po inicializaci všech částí žádnou další práci. Po přípravě všech objektů a nastartování samostatných vláken tedy začne samo sbírat data.

Stejně jako u původního řešení bude v cyklu postupně procházet přes jednotlivé položky s nastavenou historií (u ostatních to není třeba) a načítat data z přiřazených zdrojů. Po tomto cyklu budeme vlákno na sekundu uspávat, čímž budeme zajišťovat zmíněné odstupy pro evidenci hodnot do historie.

V praxi nebudou časy mezi měřeními stejně dlouhé a budou závislé na aktuální zátěži procesoru. To ale nebude příliš vadit, protože sbíraná data slouží pouze pro celkový přehled. Při produkčním nasazení je pak vhodné využívat řešení pro centralizované sledování a důležitá je pak hlavně možnost přístupu k aktuálním hodnotám položek, což zajištěno je.

2.4.7 Bezpečný přístup

S přístupem k datům ve sdílené paměti bude opět souviset bezpečnost při paralelním přístupu. Budeme muset položky nějak efektivně zamykat.

Původní řešení zamykalo celou tabulku a ukázalo se jako dostatečné. Při výhledovém možném použití na více vláknovém systému by to ale snadno mohlo vést na vznik častých konfliktů.

Pro konfiguraci jsme už řešili problém Readers-Writers a zde bychom jej mohli využít znovu na úrovni položek. I zde přitom očekáváme preferenci zápisů, aby probíhala aktualizace dat co možná nejrychleji.

Nevýhodou tohoto řešení však je, že implementace vyžaduje větší množství paměti, což by při velkém množství položek mohlo představovat problém. Při realizaci jej zkusíme využít a v případě problémů řešení v tomto směru zjednodušíme a zjednodušíme jej na původní podobu[2].

2.4.8 Sledování

Sbíraná data bude třeba kontrolovat a při splnění definovaných podmínek spouštět zadané akce podle požadavků F-P-D. Musíme tedy rozhodnout, jak budou kontroly probíhat, jak se budou zapisovat a vyhodnocovat potřebné výrazy, a jak spouštět definované akce.

[2]Testování ukázalo, že paměti je dostatek a konečná implementace používá Readers-Writers s preferencí zápisů.

Základem bude související uživatelská část konfigurace, kam bude třeba všechny informace uložit. Uvažujeme přitom kontroly, zapisovatelné jednoduchými matematickými výrazy, kde máme již připravený formát kvůli počítaným datovým položkám (zdroj Math) a vybrané akce, kde je vyžadován SNMP trap.

Příkladem takové kontroly by mohla být kontrola hraniční hodnoty průměru teploty za poslední přibližně tři minuty. Využijeme toho, že měření probíhá po uspání na jednu sekundu a průměrování probíhá po šedesáti měřeních a v historii tedy máme k dispozici zprůměrované hodnoty po přibližně minutách a máme již připravenou implementaci výpočtů z datového zdroje Math.

Budeme-li pak mít teplotu pod položkou 1.8, budeme zapisovat výsledný výraz jako:

```
(($1.8.1.0$ + $1.8.1.1$ + $1.8.1.2$) / 3) > 50
```

Je nutné podotknout, že ve výrazu počítáme průměr z agregovaných minut a průměr je tedy alespoň jednu minutu starý. Pro orientační přehled by to ale mělo být dostačující a případně je možné jej zpřesnit a používat pouze několik hodnot z posledních měření na nulté úrovni.

Uspávání mezi měřeními nezajistí vysokou přesnost z hlediska času a bylo by možné jej různými metodami zlepšit. Mohli bychom například uspávat na výrazně kratší interval a ukládat si čas, po kterém již měření musí proběhnout. Data z měření mají ale sloužit pouze pro orientační účely a i takto bude řešení dostačující.

Při splnění potřebujeme zaslat SNMP trap. Pro něj máme připravenou strukturu konfigurace a stačí tedy doplnit patřičné údaje. Z hlediska nastavení je třeba mít vybrané jeho OID, typ hodnoty, vlastní hodnotu a autentizační údaje pro správné doručení. Pro odeslání pak můžeme využít přímo nástroje snmptrap z NetSNMP.

Z hlediska údajů se pak liší parametry podle typu zabezpečení. Pro SNMP V3 s ověřením uživatele a šifrováním bychom například mohli použít:

- cíl: 10.0.0.1
- verze: 3 (SNMP V3)
- zabezpečení authPriv (autentizace a šifrování)
- uživatel: user
- autentizace: MD5
- - heslo: userPassword
- šifrování: DES
- - heslo: encPassword

Při splnění podmínky bychom takto mohli zbytečně zatěžovat server i zařízení jeho opakovaným zasíláním a bude dobré zvážit četnost opakování kontrol. Zavedeme tedy parametry pro maximální počet opakovaného spuštění v řadě a minimální čas v sekundách mezi opakováními.

Ne každou kontrolu je navíc potřeba provádět po každé aktualizaci dat. Zavedeme tedy také minimální čas mezi opakováními v sekundách a budeme pamatovat poslední čas kontroly.

2.5 Aktualizace firmware

Dále se podíváme na stav aktualizace firmwaru a možnosti dokončení podle návrhu [2]. V původním stavu byly možnosti pokračování velmi omezené, protože nebylo možné jednoduše začít upravovat řídící modul a podle toho byl i vytvořen návrh.

To teď již ale nebude platit a návrh tomu můžeme přizpůsobit. Původně se počítalo se dvěma procesy. Jednomu by se předávaly požadavky z webového rozhraní a druhý by řešil případný zápis balíčků do zařízení.

Tvorbu balíčků a jejich příjem budeme moci přidat jako součást přímo do webového rozhraní. Ta totiž neohrožuje systém ani případný průběh aktualizace, protože balíček před ním bude ještě zkontrolován, ušetříme přitom ale proces a paměť, kterou by proces vyžadoval.

Aktualizace firmwaru je však velmi citlivá a jeho průběh tedy v samostatném programu bude muset zůstat. V průběhu aktualizace navíc bude nutné balíček vybalit a zkontrolovat jej, což bude vyžadovat další paměť, kterou již ale na našem zařízení nemáme k dispozici. Bude tedy vhodné po úspěšném přijetí balíčku jej uložit na pevně dané místo a ukončit řídící modul.

To necháme detekovat startovací skript, který se jej pokusí znovu zapnout. Před jeho start zapnutí zapisujícího aktualizačního procesu. Ten se po startu pokusí balíček z umístění vzít, zkontrolovat a nahrát. Pokud aktualizace proběhne, program vyvolá restart systému a spustí se s novou verzí, jinak program skončí a bude se pokračovat startem řídícího modulu.

Úplná realizace aktualizace ale zůstane pouze formou návrhu, zápis firmwaru je totiž velmi riskantní a vyžadoval by další spolupráci s firmou SVM s.r.o. kvůli nutnosti úpravy zavaděče. V rámci této práce tedy realizaci aktualizačního procesu zakončíme u úspěšného přijetí balíčku s novým firmwarem.

2.6 Přístupová rozhraní

Dále se již podíváme na návrh realizace přístupových rozhraní. Ta by nám měla umožnit ovlivňovat chování zařízení a přistupovat ke sbíraným datům. Zatím budeme přitom potřebovat protokoly HTTP a SNMP, časem by ale teoreticky mohla být potřeba další. Budeme se přitom snažit zavést nějakou pro ně obecnější strukturu na úrovni modulu.

V obou případech půjde prakticky o samostatné části, kde obě budou vyžadovat přístup k systému (objekt Machine), odstartování a zastavení pro korektní ukončení. Podle toho tedy zkusíme navrhnout možnost tvorby takových rozšíření.

Vytvoříme třídu, která bude fungovat jako registr takových rozšíření. Každé pak bude muset implementovat objekt s daným rozhraním a být v kódu zaregistrováno. Všechna rozšíření pak umožníme vždy zapnout a vypnout, což povede na postupné zavolání této akce na jejich objektech.

Na úrovni zdrojového kódu rozšíření oddělíme do samostatného sub modulu pod „Machine“ pro zvýšení přehlednosti.

2.6.1 SNMP

Začneme se SNMP. To by nám mělo posloužit hlavně pro účely sledování a data tedy musí být možné hlavně číst. Tam, kde to lze, by ale bylo dobré umožnit i jejich zápis.

Pro realizaci použijeme doporučenou knihovnu NetSNMP. Bude přitom potřeba udržovat funkční hlavní proces SNMP a vlákno v našem programu, které bude obsluhovat požadavky na naši část tabulky.

2.6.1.1 Správa oddělených procesů

Hlavní proces SNMP bude nutné zapínat a vypínat pro korektní ukončení. Kromě toho by se nám hodilo jej průběžně kontrolovat a v případě nečekaného pádu provést jeho restart.

Zavedeme tedy další vlákno, které bude tyto akce zajišťovat. Budeme jej označovat jako „supervisor“ a bude u něj možné zaregistrovat podobné funkční třídy, u nichž bude pravidelně volat kontrolní funkci a případně restart.

Zároveň bude takto zaregistrované objekty poskytovat přes textové identifikátory. Jinde v kódu tak bude možné si takový objekt vytáhnout a proces restartovat. Tak budeme schopni například schopni přinutit proces SNMP k načtení nové konfigurace po aktualizaci.

2.6.1.2 Komunikace se snmpd

Komunikace s NetSNMP procesem prostřednictvím protokolu AgentX bohužel není zcela ideálně zdokumentovaná. Zkusíme navíc využít informací ze zdrojů [26] [27] [28] a přijít s použitelným řešením.

Ukazuje se, že bude třeba implementovat obslužnou funkci pro určitou počáteční část identifikátoru (OID) v MIB. Tu pak zaregistrujeme u hlavního procesu SNMP a ten nám bude patřičné dotazy předávat ke zpracování. Základem pak bude v podstatě nekonečný cyklus, který bude vyvolávat čekání na komunikaci a případně obdržené dotazy může zpracovat a výsledky opět předat hlavnímu procesu přes knihovní funkce.

2.6.1.3 Zpracování dotazu

Z hlediska dotazů se zaměříme na ty od klientských programů. Budeme tedy implementovat dotazy na čtení (GET, GET NEXT, GET BULK) a zápis (SET) [29].

- **GET**

  Základním dotazem je GET pro získání dat, kde dotazování probíhá přes OID. Informace o těchto identifikátorech a souvisejících položkách již máme díky konfiguraci struktury a databázi. Budeme tedy prakticky muset pouze vyhledávat položky přes textovou podobu části OID v evidenci, kterou si uložíme formou tabulky.

  Pokud uspějeme, vyčteme data a vrátíme je. K tomu bude třeba znát datový typ v rámci SNMP. Doplníme tedy do položek informaci o SNMP typu a metodu, která nám ji bude poskytovat.

  Problém ale budeme mít u typů, které nejsou součástí standardního SNMP, jako jsou typy s plovoucí řádovou čárkou.

  To lze řešit různými způsoby. Můžeme provádět převod do celočíselných typů, a předávat je tedy v jiné formě reprezentace pro konverzi, nebo jednoduše v textové formě. Tyto přístupy budou fungovat vždy, ale vyžadují další operace na klientovi.

  Některé nástroje, včetně těch z balíku NetSNMP, pak podporují také méně rozšířené typy, označované jako Opaque. Zde nalezneme chybějící datové typy, jako je právě Float nebo Double.

  V této práci je využijeme. Při problémech je možné implementovat další typy na úrovni databáze, které budou využívat jiný způsob konverze.

  Samotná data pak budeme předávat po bajtech. To už nebude problém, protože v této podobě budou sbírána do naší databáze. Při implementace pouze nesmíme zapomenout na správné zamykání položek.

  Posledním problémem pak bude přístup k datům z historie. Jelikož jde o pole polí, mohli bychom je takto poskytovat i v SNMP.

  U položek s historií tedy budeme v SNMP registrovat více OID podle umístění a nastavení historie. Budeme-li mít například položku „.2“ s historií o hloubce alespoň 1, šířce alespoň 2 a interval aktualizace databáze 1 sekunda, pak hodnotu zpožděnou alespoň o 1 sekundu získáme přes identifikátor .2.0.1, kde 0 značí nejnižší úroveň, s pauzami přibližně 1 sekunda, a 1 odstup o jednu pauzu.

- **GET NEXT**

  Další dotaz lze využít při průchodu celou databází a označuje se GET NEXT. Funguje v podstatě jako dotaz GET, ale před vyhledáváním hodnoty OID dojde k jeho posunu na další v pořadí.

K posunu budeme potřebovat udržovat setříděnou databázi našich OID, ve které jej budeme schopni najít. Zavedeme tedy pole, do kterého při inicializaci všechna OID vložíme ve správném pořadí podle čísel v tečkové notaci. Následně implementujeme vyhledávání binárním půlením, čímž zajistíme dostatečnou rychlost.

Takovéto procházení by mohlo ovšem přinášet bezpečnostní riziko. Dotazovací program by se mohl dostat i k hodnotě, ke které nemá přístupová práva přes konfiguraci SNMP, protože na této úrovni je neověřujeme. Abychom toto mohli řešit, museli bychom přistupovat přímo k funkcím hlavního agenta a dotazovat se ho na práva k objektům podle aktuálního nastavení.

V NetSNMP je to řešeno tak, že se u odpovědi testuje, zda na ni uživatel má právo a pokud ne, dotaz se opakuje pro další položku. Tím je tedy bezpečnost zajištěna, bohužel to má ale negativní vliv na rychlost.

Pokud budeme mít rozsáhlou tabulku s mnoha vyhrazenými sekcemi, budou při procházení u omezených klientů hrozit vypršení časových limitů. S tím musíme počítat při navrhování naší tabulky a mít je pokud možno co nejmenší nebo umístěné v nějaké jiné části hierarchie, u které nebude standardně důvod jí procházet.

- **GET BULK**

  Od SNMP V2 existuje také dotaz GET BULK [29]. Ten je ale v NetSNMP prakticky překládán na sérii požadavků GET NEXT, takže jej nemusíme zvlášť implementovat.

- **SET**

  Posledním dotazem bude SET, který nastavuje zaslanou hodnotu na zadaném OID. Základní vyhledání položky bude stejné, jako u GET, musíme ale položku zamykat pro zápis.

  Navíc bude třeba řešit případnou chybu při zápisu. Ta může vzniknout například kvůli nemožnosti zápis provést, zadáním špatných dat nebo jiné chybě například v průběhu zapisování kvůli specifickým vlastnostem zdroje. NetSNMP s tím počítá a umožňuje klienta informovat i o chybě voláním patřičných funkcí.

2.6.2 HTTP

Dále se zaměříme na webové rozhraní. Zde přitom není nutné, aby byla nadále funkční jeho původní verze. Naopak připouštíme možnost, že jej implementujeme znovu a lépe tak, aby bylo snáze rozšiřitelné a i vlastní rozhraní pro uživatele se znovu implementuje.

2.6.2.1 Architektura

Bylo by dobré jej připravit tak, aby nad ním bylo možné postavit moderní webovou aplikaci, například v jazyce JavaScript (N-N-H-01). Místo původního řešení bychom mohli generovat pouze odpovědi ve formátu JSON, který lze snadno zpracovávat různými jazyky, a webové rozhraní by mohlo představovat klienta, který je pouze servírován z paměti pojítka.

Pokud se úplně oprostíme od původního řešení, mohli bychom mírně změnit architekturu. Mohli bychom mít webový proces, který by dokázal generovat celé odpovědi aplikačního rozhraní a poskytovat soubory uživatelského. Tím bychom se vyhnuli nutnosti opakovaně spouštět CGI skript a nepotřebovali bychom ani původní webový server.

Komunikaci přes HTTP bychom ale stále museli řešit a implementovat vlastní program, který by původní webový server nahrazoval. Pracoval by také po otevření potřebných portů pod jiným než superuživatelem, přijímal a zpracovávat požadavky a případné získávání dat a provádění požadovaných akcí řešil prostřednictvím rozhraní mezi ním a řídícím modulem podobně, jako u původního CGI skriptu.

Bylo by dobré pro implementaci webové části použít nějakou volně dostupnou knihovnu. Hledání ukázalo, že jich je více. Jako nejvhodnější vypadá libmicrohttpd[30], která je dostupná přímo i v balíku Buildroot a stačí ji tedy přidat do sestavování.

Podporuje přitom i funkce pro zabezpečení pomocí SSL a základní mechanismy, jako je autentizace pomocí HTTP Basic nebo Digest. Zároveň nám umožňuje vybrat mezi obsluhou vlákny nebo voláním select[3], což bude vzhledem k jednomu jednojádrovému procesoru vhodnou volbou.

Její licence zároveň zcela vyhovuje tomuto použití. Je pod LGPL a můžeme proti ní tedy kompilovat a dynamicky linkovat.

Při realizaci ji tedy využijeme a vydáme se touto cestou. Na ukázku zároveň přiložíme jednoduchou webovou aplikaci, která umožní vizualizovat některé sbírané informace, na dokázání použitelnosti.

2.6.2.2 Bezpečnost

Dále se více zaměříme na bezpečnost komunikace se zařízením přes webové rozhraní. Nemáme zatím žádné požadavky na více uživatelských účtů nebo jejich role, připravíme tedy systém zatím s jedním správcovským účtem, chráněným heslem, a další rozšíření ponecháme na pozdějším rozvoji.

Z hlediska bezpečnosti HTTP provozu je standardem použití SSL. To knihovna podporuje, bude k tomu nutné ale přidat do firmwaru knihovnu libgnutls [31]. Tu můžeme přidat v menu Buildroot, není ale jasné, kolik bude potřebovat místa. Při problémech budeme muset zvážit, zda je možné ji v

[3] linuxové sokety a select: http://linux.die.net/man/2/select

systému ponechat a případně zvolit jinou implementaci nebo absenci SSL. Předběžné testy ukazují, že místa bude dostatek.

Komunikace prostřednictvím SSL bude vyžadovat klíč a certifikát. Využívání skutečně ověřených podepsaných certifikátů by bylo ale příliš nákladné a i podepisování firemním by zatím bylo komplikované, protože by bylo nutné řešit také certifikační autoritu.

Využijeme zde tedy zatím pouze sebou podepsané certifikáty. Pro zvýšení bezpečnosti budeme certifikát generovat při prvním spuštění HTTP rozšíření a uložíme jej trvale do uživatelské konfigurace ve flash paměti.

Jako platnost certifikátu nastavíme deset let. Do té doby bude určitě nutné provést změny, které si vyžádají aktualizaci firmwaru a s vysokou pravděpodobností povedou i k aktualizaci konfigurace a nutnosti vyplnit ji znovu. Jinak bude po deseti letech teoreticky nutné zařízení resetovat.

U certifikátu budeme muset zvážit také velikost klíče. Na běžných serverech bychom dnes asi použili velikost 4096 bitů. To ale zvyšuje také výpočetní náročnost, 1024 bitů se zase již dnes ukazuje jako nedostatečných [32]. Jako základní tedy zvolíme 2048 bitů a volbu případně upravíme podle výsledků z testů výkonnosti.

Samotné ověření uživatele pak lze řešit mnoha dalšími způsoby. Kromě standardních mechanizmů, jako je HTTP Basic a Digest lze implementovat také nějaký další, například s využitím tokenů[4], ale tyto techniky bychom při stávajících potřebách nevyužili a stálo by nás to další výpočetní výkon.

Zatím se tedy spokojíme s HTTP Basic autentizací přes SSL.

2.6.2.3 Formát komunikace

Původní řešení generovalo přímo webové stránky v HTML pomocí CGI skriptů. V rámci nové implementace bychom ale rádi data předávaly ve formě JSON. Zápis ve formátu JSON zvládneme poměrně snadno, hodila by se nám ale knihovna, která nám formát bude zpracovávat z požadavků při jejich přijetí.

Takových knihoven je opět mnoho. Po zkoušení se jako přijatelná ukazuje JSONXX[33], která je opět pod dostatečně volnou licencí a nabízí použitelné rozhraní pro práci s různými datovými typy.

2.6.2.4 Akce rozhraní

Dále se více zaměříme na samotné akce, které bude rozhraní muset implementovat. Budeme je identifikovat HTTP metodou a částí adresy, na kterou bude třeba požadavek zaslat.

Často budeme řešit správu. Tou se bude myslet provádění základních operací, jako je čtení, přidání, úprava nebo smazání.

Požadavky budeme přitom muset vnitřně nějak vhodně reprezentovat a ten správný volit. Zavedeme tedy pro každý objekt (Router) a zaregistrujeme je v

[4]například WSSE: http://en.wikipedia.org/wiki/WS-Security

něm do tabulky spolu s HTTP metodou (GET, POST, PUT, DELETE, ...). Pro případné proměnné v adrese zavedeme notaci „{promenna}“, která bude umožňovat nastavit hodnotu mezi oddělovači v adrese nebo z konce adresy do tabulky proměnných. Načtené proměnné budeme předávat akcím při jejich volání.

Zpracování úkonů bude často vyžadovat výměnu dat s hlavním procesem řídícího modulu. K tomu implementujeme lokální rozhraní (Manager), které nám to umožní, jako u původního řešení s CGI. Oproti němu si ale implementaci zjednodušíme a odpovědi budeme již pak generovat v celé textové podobě také v něm.

Konverzi dat tak budeme muset řešit pouze při příjmu požadavků od uživatelů, což je ale žádoucí, protože zde se mohou objevit škodlivá data. Pro lokální rozhraní budeme data předzpracovávat do podoby číselných identifikátorů, za kterými umožníme zasílat hodnoty, kde bude vždy typ, případně doplňující informace a vlastní data.

- **Obecné**

  Základem bude obecně schopnost ověřit, zda je rozhraní funkční. Doplníme tedy volání, na které se bude zařízení při úspěšném přihlášení alespoň hlásit nějakou standardní odpovědí.

  K tomu zařadíme také možnost měnit nastavení rozhraní. Zde tedy hlavně údaje pro HTTP Basic autentizaci.

  Budeme také muset poskytovat soubory. Zavedeme tedy pevný adresář, ze kterého bude možné soubory stahovat.

  Zde bude třeba akci dobře zabezpečit. K tomu využijeme funkce „realpath“ [34], která by měla být při správném použití bezpečná a převádí zadanou cestu na skutečnou, takže by ve výsledné cestě mělo stačit zkontrolovat předponu.

  Někdy by mohlo být dobré mít možnost zařízení vzdáleně restartovat. Přidáme sem tedy také tuto možnost. Provedení restartu bohužel ale vždy povede ke ztrátě dat v dočasných pamětech a může být zneužito pro omezení funkčnosti celého zařízení jeho opakovaným voláním na dobu restartu. Správci tedy nesmí uniknout přihlašovací údaje.

- **Data**

  Důležitý je pak přístup k datům ve vnitřní databázi. Zpracujeme tedy metody pro získávání informací o datových položkách, jejich struktuře a získávání nasbíraných hodnot.

  Pro kompletnost doplníme také nastavování položek. Informace budeme přijímat v textové podobě a na úrovni položky se tedy pak bude muset řešit konverze podle jejího skutečného typu.

- **Sledování**

  Bude se také hodit mít možnost nastavit sledování a k tomu tedy připravíme akce pro správu kontrol a následných operací.

  Z operací budeme implementovat pouze SNMP trap a u nich jsou třeba přihlašovací údaje na cílový server. Jelikož se budou pravděpodobně opakovat, budeme je ukládat a spravovat samostatně a z operací se na ně pouze odkazovat.

- **SNMP**

  Kromě takových notifikací bude třeba také mít možnost upravovat konfiguraci samotné hlavní SNMP služby. Není ale žádoucí implementovat obalující volání pro všechny jeho konfigurační direktivy.

  Místo toho tedy umožníme administrátorovi zasílat přes webové rozhraní celý konfigurační soubor. Ve webovém rozhraní pak bude dobré implementovat pomocný nástroj, který umožní základní direktivy zadat a konfiguraci za uživatele vygeneruje. Editace přímo v textu bude pak nutná pouze pro pokročilejší nastavení.

- **Zařízení**

  Dále bude třeba umožnit konfiguraci zařízení. Tak bychom mohli umožnit změnu nastavení Ethernetového rozhraní pojítka a později případně i další.

  Toto nastavování se ale bude velmi lišit podle zařízení a vyžadovat přidané další kontroly. Každý objekt zařízení tedy bude implementovat aktualizační metodu. Té bude možné předat nastavení formou tabulky, ze které si vybere známé hodnoty a podle nich a výsledků kontrol upraví své nastavení a případně vše uloží trvale do uživatelské konfigurace.

- **Aktualizace zařízení**

  Spadají sem také akce související s aktualizací firmwaru zařízení. Z pohledu webového rozhraní bude třeba implementovat dvě základní části procesu.

  První je stažení aktualizačního balíčku. Ten bude obsahovat všechny potřebné informace podle návrhu a bude šifrovaný. K tomu využijeme volání z knihovny, které doimplementujeme.

  Druhá akce pak umožní nahrát aktualizační balíček. Budeme tedy muset řešit nahrávání souboru, dešifrování a kontrolu. Po analýze už víme, že samotná aktualizace nebude možná, balíček ale musíme být schopni alespoň uložit do paměti a informovat uživatele. Zároveň doimplementujeme do knihovny jeho ověřování a při pozdějším dokončování aktualizačního procesu tak již bude stačit ji použít.

Výsledný návrh akcí je v příloze C.

2.7 Struktura modulů

Vytvořené části rozdělíme do složitější struktury. Kód rozdělíme podle původního základního návrhu do složky „mod“ podle závislostí. Výsledkem bude následující struktura.

- `conf` ... správa konfigurace pro SW a zařízení
- `debug` ... ladící a informační hlášky
- `gzip` ... komprese dat
- `json` ... práce s daty ve formátu JSON
- `locker` ... zamykání na způsob Readers-Writers
- `machine` ... hlavní sub modul „Machine“
- `snmp` ... správa informací k SNMP
- `string` ... práce s texty
- `xml` ... správa XML

Je vidět, že máme prakticky hlavní zdrojové kódy modulu (machine) a k nim různé další pomocné části. Většinu z nich bude možné používat i samostatně.

Do hlavního modulu systému pak přijdou implementace jednotlivých funkcionalit. Jeho struktura bude následující.

- `check` ... kontrola sbíraných dat
  - `operation` ... dostupné akce
- `conf` ... správa uživatelské konfigurace
- `data` ... databáze dat
  - `item` ... implementace položek různých typů
  - `source` ... datové zdroje pro položky
- `device` ... obslužné třídy pro zařízení v systému
  - `bus` ... komunikace přes sběrnice
  - `net` ... komunikace po síti
  - `storage` ... datová úložiště
- `extensions` ... samostatná rozšíření softwaru
  - `manager` ... implementace lokálního spravujícího rozhraní
    - `req` ... obsluhy jednotlivých požadavků
  - `snmp` ... implementace klientské části SNMP rozhraní
- `supervisor` ... sledování samostatných procesů

K tomu přibude také samostatná struktura, kde budou zdrojové kódy odděleného webového rozhraní. Ty umístíme do samostatné složky „http“ mimo moduly a vystačíme si pouze s podsložkou „request“ pro implementované akce.

2.8 Hlavní program

Nakonec se podíváme na životní cyklus řídícího modulu jako celku.

2.8.1 Údržba záznamů

První bude údržba záznamů z fungování (log). Ten je důležitý, protože je mimo jiné i případnou součástí aktualizačního balíčku a servis z něj může zjistit případné problémy a nesrovnalosti.

Musí mít ale omezenou velikost, aby nevyčerpal paměť zařízení. Při spouštění tedy umožníme zadat cestu k němu a maximální počet záznamů a podle toho pak budeme průběžně automaticky jeho obsah průběžně promazávat při zápisech přes ladící třídu.

2.8.2 Konfigurace

Modul navíc vyžaduje složitější konfiguraci v podobě XML souborů. Na její obsah byl po celou dobu návrhu kladen velký důraz a bude muset být načtena z nějakého umístění. Přidáme při spouštění tedy také parametr pro cestu k hlavní konfiguraci.

2.8.3 Automatické restartování

Při práci řídícího modulu může teoreticky docházet k chybám, které by mohly vést až k jeho náhlému ukončení. Mohlo by jít jak o chybu programu, tak i o nečekané selhání nějakého hardwarového prvku zařízení.

Pro menší chyby, které nezpůsobí havárii, zavedeme vnitřní proces restartování. Při spouštění modulu budeme udávat parametry pro dobu inicializace a pauzu mezi restartováními. Pokud na úrovni programu dojde k ukončení jeho funkce, na chvíli proces uspíme a znovu spustíme. Pokud došlo k ukončení v čase pro inicializaci, pak zároveň prodloužíme pauzu. Když k pádu dojde až po inicializaci, pauzu obnovíme na původní hodnotu.

Stávající implementace by pro tyto případy však měla být odolná a hrozí spíš úplný pád procesu v důsledku nějaké vážnější chyby například při práci s pamětí, kterou ani intenzivním testováním nelze zcela vyloučit. Vytvoříme tedy startovací skript tak, aby se případně program po menší pauze sám znovu zapnul.

K ukončení programu může také dojít kvůli požadavku na aktualizaci. Na toto místo tedy později přijde také spouštění aktualizačního programu, který se bude pokoušet nahrát firmware z definovaného umístění.

2.8.4 Životní cyklus

Po tomto rozvržení už můžeme přejít k vlastnímu fungování modulu. Při spouštění na začátku načte svoje základní parametry a následně:

- načte hlavní konfiguraci,
- inicializuje objekty zařízení,
- načte uživatelskou konfiguraci,
- donastaví podle uživatelské konfigurace zařízení,
- inicializuje databázi pro sbíraná data,
- připraví pomocné objekty pro SNMP a zasílání notifikací,
- nastaví kontroly sbíraných dat,
- spustí rozšíření řídícího modulu,
- začne sbírat data ze zařízení

Pro ukončení pak zaregistrujeme na signály SIGINT a SIGTERM volání ukončující funkce hlavního modulu [35][36]. Ten pak postupně informuje všechny části o ukončení, což povede k ukončení funkce samostatných vláken a procesů a následně se ukončí i celý program.

2.8.5 Schéma

Výsledný návrh tvoří poměrně komplexní řešení a jeho architekturu ještě pro přehlednost znázorňuje také obrázek 2.3. Rozděluje všechny části do tří hlavních bloků.

Nejvýše je vnější síť, do které je přistupováno přes na desce dostupné Ethernetové rozhraní. Z ní je přístupné rozhraní SNMP a web.

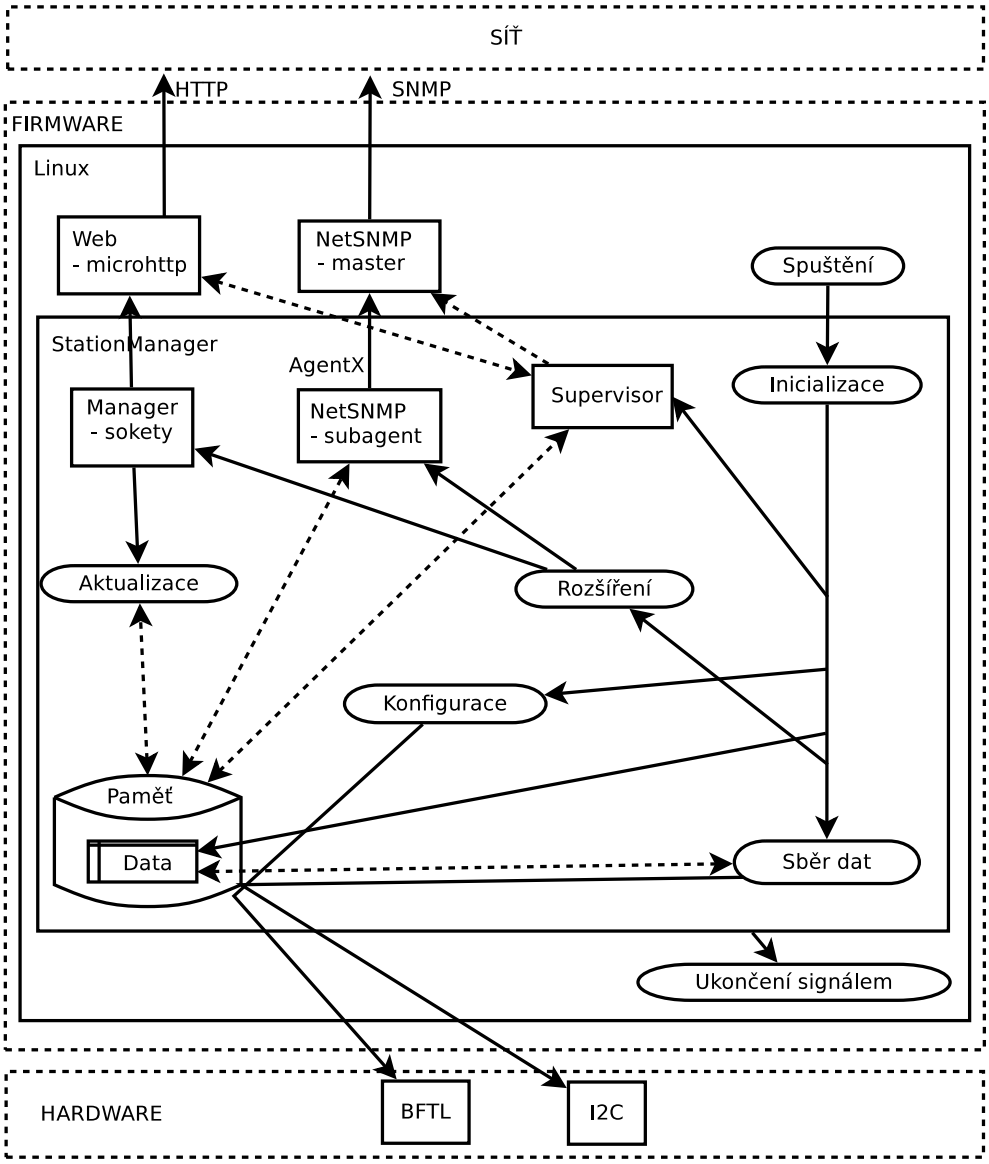
Na nejnižší úrovni je pak samotný hardware zařízení. Klíčová je flash paměť, ke které přistupujeme prostřednictvím ovladače BFTL a I^2C sběrnice, kterou zařízení ovládáme a získáváme potřebná data.

Uprostřed pak stojí samotný firmware, založený na OS Linux. Zde pracuje náš řídící modul, označený jako StationManager.

K zapnutí modulu dochází inicializačním skriptem. Na začátku jsou zpracovány parametry programu, podle kterých je vybrán konfigurační soubor. Následuje příprava supervisoru, jsou samostatně spuštěna rozšíření a hlavní vlákno začne sbírat data. Rozšíření připraví pomocná vlákna a spustí procesy pro obsluhu vnějších rozhraní. Tím je vše připraveno k práci.

V případě potřeby je možné provést aktualizaci. Ta povede k ukončení modulu a jeho novému spuštění inicializačním skriptem buď přímo, nebo po restartu pojítka, vyvolaném po případné úspěšné aktualizaci.

K ukončení modulu může jinak dojít také přímým zasláním některého systémového signálu nebo chybou systému.



Obrázek 2.3: Schéma návrhu

Kapitola 3

Implementace

V rámci implementace se zaměříme na přidání do stávajícího sestavovacího prostředí pro firmware pojítka a vyřešení dalších možných problémů. Zároveň si zhodnotíme úspěšnost realizace návrhu a obecnost pro případné použití na dalších zařízeních.

3.1 Flash paměť

Základem bylo zajištění podpory BFTL. To ovšem vyžadovalo použití novější verze bootloaderu s jeho podporou, jinak nedocházelo ke zpřístupnění zařízení OS Linux pro jeho jednotlivé alokované sektory.

K desce přitom ale nebylo k dispozici zařízení pro její programování. Po dohodě s SVM s.r.o. byla ale deska přeprogramována s použitím dodané upravené verze bootloaderu a ovladač bylo nakonec možné používat.

3.2 Doplnění závislostí

Pro sestavování firmwaru bylo nutné doplnit knihovny a upravit některá nastavení procesu sestavování.

Základem byla podpora vláken a výlučných zámků. Původní firmware používal uClibc ve verzi 0.9.31, jehož součástí ještě nebyla podpora pro verzi NPTL[19] na platformě SH4, ale používal pouze původní implementaci LinuxThreads. Použití NPTL bylo ale v době psaní práce na OS Linux bylo poměrně standardní a uClibc ji už přitom v novějších verzích na SH4 podporovalo. Bylo tedy aktualizováno na novější verzi 0.9.33.2 a zkompilováno s ním.

Implementace webového rozhraní vyžadovala knihovnu libmicrohttpd s podporou SSL. Byl tedy přidán tento balíček a také již zmíněná libgnutls.

Tak již není třeba mít součástí firmwaru původní webový server ani řídící modul a ze sestavování tedy byly odebrány. U původního modulu stačilo

odebrat jej přes konfiguraci, u webového serveru pak editací konfiguračního souboru pro BusyBox.

SNMP pak vyžadovalo doplnění knihovny NetSNMP. Tu bylo také možné přidat pouze s použitím konfiguračního nástroje.

Při následném pokusném sestavování bylo zjištěno, že pro používání knihovny pugixml bude potřeba přidat také podpora pro široké (více bajtové) znaky[5] pro podporu kódování UTF-8[6]. Tu opět stačilo pouze přidat v konfigurační nabídce menu Buildroot a firmware pak již bylo možné úspěšně sestavit.

3.3 Testovací prostředí

K vývoji a testování byl k dispozici virtuální stroj. Aby na něm bylo možné testovat, bylo třeba jej doplnit o další nástroje a knihovny.

Základem bylo NetSNMP a libmicrohttpd. NetSNMP stačilo použít přímo z balíčků systému, libmicrohttpd pak vyžadovalo také libgnutls a kompilaci novější verze s podporou SSL.

Do systému pak byly doplněny některé složky a specifická varianta konfigurace, aby nebylo nutné mít k dispozici například sběrnici I^2C. K tomu postačilo doplnit objekt sběrnice o simulační režim, kdy neprovádí zápisy a místo čtení vrací náhodná data.

Z hlediska BFTL pak zápis probíhá prakticky standardním způsobem. Stačilo tedy poskytnout umístění souboru a data konfigurace již pak zůstávala uložena standardně na disku. Spouštění pak bylo možné provádět po kompilaci jednoduše s přidáním parametru s cestou ke zkomprimovanému konfiguračnímu souboru, který lze generovat z jeho XML podoby přiloženým skriptem.

3.4 Bezpečnost

Za dobu psaní práce se objevila řada bezpečnostních chyb, které se dotkly i použitého systému a knihoven. Z těch mediálně propagovaných byl zajímavý například ShellShock [37] nebo The Heartbleed Bug [38].

ShellShock se týkal chyby v programu bash, který je často využíván dalšími nástroji při volání programů. Do něj bylo možné relativně snadno dostat cizí kód patřičným nastavením hlaviček komunikace u některých protokolů. To se týkalo nejen použití CGI skriptů na webovém serveru, ale i třeba komunikace přes SSH[39]. Bash ale nebyl součástí firmwaru a ani testovací příkazy neukázaly, že by touto chybou zařízení chybělo.

Heartbleed se pak týkal chyby v OpenSSL. Chyba umožňovala přístup přímo do paměti zařízení, což by mohlo víc než jen ohrozit implementaci licenční politiky na zařízeních.

[5] široký znak (wide char): http://en.wikipedia.org/wiki/Wide_character
[6] UTF-8: http://en.wikipedia.org/wiki/UTF-8

Tuto knihovnu přitom zařízení využívá a byla tedy aktualizována. Pro otestování pak byl použit jeden z veřejně dostupných skriptů[7].

Více se na bezpečnost podíváme v samostatné kapitole.

3.5 Porovnání náročnosti na prostředky

Při implementaci bylo nutné změnit řadu nastavení, přidat další software a knihovny. Pro srovnání bude dobré se podívat, jak se to projevilo na velikosti výsledného firmwaru a zda nám na zařízení zůstane po úplném startu dostatek volné paměti pro další činnosti.

Nový firmware narostl oproti původní na 5,3 MB. To je stále ve stanoveném limitu sedmi megabajtů [3] a neměl by to tedy být problém.

Další programy a knihovny stojí ale také paměť RAM. Kritická situace přitom nastává při procesu aktualizace firmwaru, kdy se do paměti nahrává aktualizační balíček. S tím bylo ale počítáno a proces byl podle toho navrhnut. Testování prokázalo, že systém si i s tímto krizovým momentem poradí a nedojde k vyčerpání prostředků.

3.6 Podporovaná zařízení

Pro další užívání bude dobré také zvážit možné použití na dalších zařízeních. Při testování na platformách SH4[8], i386[9] ani x86[10] nebyly zaznamenány žádné problémy a prakticky by tedy mělo být možné modul provozovat i jinde při splnění jeho hlavních závislostí a dostatku zdrojů.

Samotné cílové zařízení by pak nemělo mít výrazně slabší procesor ani menší paměť, než má použité pojítko SkyWire. Jinak bude nejspíš třeba implementaci upravit a je pak možné zvážit například odebrání podpory SNMP nebo SSL u webového rozhraní.

3.7 Webové rozhraní

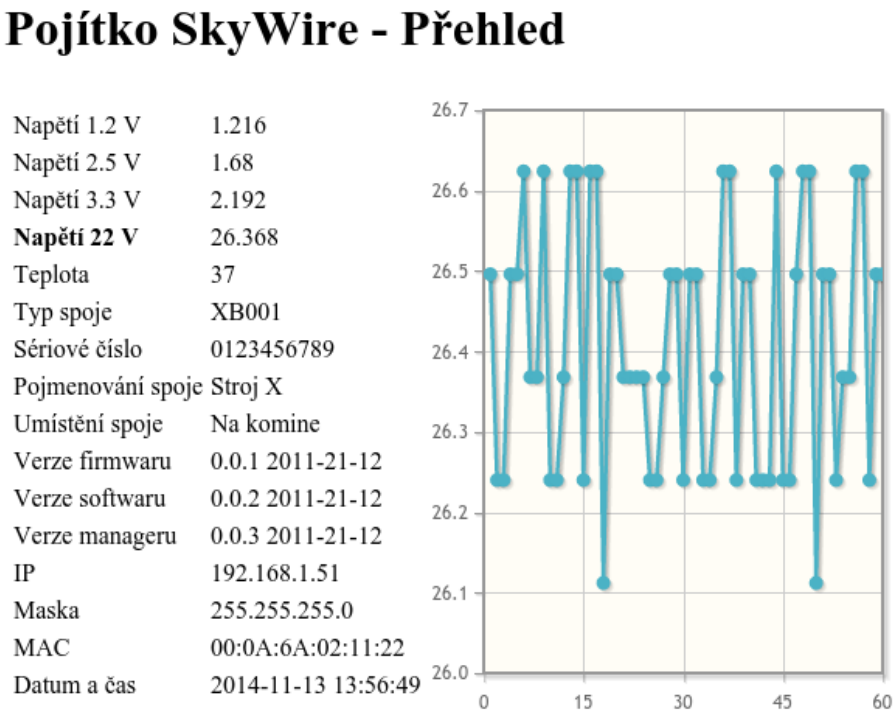
Na otestování byla v rámci vývoje aplikačního HTTP rozhraní vytvořena jednoduchá webová aplikace pro zobrazení některých sbíraných dat. Její ukázka je na obrázku 3.1.

[7] test na chybu Heartbleed v OpenSSL:
https://github.com/noxxi/p5-scripts/blob/master/check-ssl-heartbleed.pl

[8] platforma SuperH: http://en.wikipedia.org/wiki/SuperH

[9] platforma i386: http://en.wikipedia.org/wiki/Intel_80386

[10] platforma x86: http://en.wikipedia.org/wiki/X86-64



Obrázek 3.1: Ukázka nového webového rozhraní

Kapitola 4

Bezpečnost

Při realizaci byl kladen velký důraz na bezpečnost, ne vše ale dokážeme vždy zcela podchytit. V této kapitole se podíváme na rizika, která s sebou nese použití vzniklého modulu.

4.1 Základní rozdělení problémů

Rozdělíme si problémy do několika hlavních kategorií.

První jsou z vnějšího prostředí, které ohrožují uživatele zařízení. Jeho zařízení totiž může přestat fungovat tak, jak si představoval a nemusí být nadále schopno plnit svou funkci. Mohli bychom je zařadit mezi vnější útoky.

Do druhé by mohli patřit naopak ty, které od uživatele zařízení přicházejí. Zamýšlená licenční politika totiž omezuje funkčnost jinak výkonného zařízení a uživatel by mohl chtít zneužít nějakého nedostatku a ovlivnit software tak, aby licenční omezení zmírnil nebo zcela odstranil. Mohli bychom je označit za vnitřní útoky.

Do třetí pak můžeme zařadit problémy, spojené s chybami v softwaru, které přitom ale ani nemusí být možné napadnout zvenku. Mohou souviset s neplánovanou souhrou událostí, které povedou k nečekaným pádům nebo jinému poškození, například informací v pamětech, nebo jinak naruší fungování hardwaru zařízení.

Kromě zde vyvíjeného softwaru je třeba z hlediska bezpečnosti také zvážit další software v rámci firmwaru a zařízení jako celek. To bylo již zmíněno i v rámci implementace, kde jsme cíleně aktualizovali některé balíčky kvůli známým chybám. Více se touto problematikou, včetně fyzické bezpečnosti zařízení, zabývá samostatná práce o bezpečnosti [3].

4.2 Chyby v softwarovém vybavení firmwaru

Začneme možnými chybami v knihovnách a balíčcích. Jak u balíčků z Buildroot, tak i knihoven modulu se budeme snažit zajistit co nejnovější a stabilní verze, abychom omezili výskyt chyb.

Při vývoji bylo třeba toto kontrolovat a aktualizovat. Šlo zejména o balíčky OpenSSL, GnuTLS, zlib, libmicrohttpd, NetSNMP nebo uClibc. U knihoven modulu pak také o PugiXML, které by mohlo obsahovat chybu spojenou se čtením informací z uživatelského XML souboru, a JSONXX, který by mohl být napadnutelný přes zvláštně upravený obsah ve tvaru JSON, zaslaném přes webové rozhraní.

Aktualizace bylo třeba provádět manuálně. V rámci balíčkovacího systému totiž zatím nemáme žádný automatizovaný proces, který by byl schopen nové verze detekovat a automaticky provádět aktualizace a vytvářet nová sestavení, a podobně tomu bylo i u knihoven modulu.

Po aktualizacích bylo nutné opět testovat. Může totiž dojít ke změnám, které povedou k novému nečekanému chování.

Z hlediska samotného provádění aktualizací firmwaru je připraven aktualizační proces, který později umožní nahrát nově sestavený firmware po síti. V ideálním případě by mohl vzniknout také aktualizační server, který by využil rozhraní a byl schopen aktualizovat automaticky i větší sítě ze zařízení tohoto typu.

To ale určitě nebude vždy možné, protože se počítá i s používáním na sítích bez přístupu k internetu. Pak bude třeba nastavit ve firmě proces, který průběžné aktualizace zajistí.

4.3 Nedostatky implementace

Další kategorie se týká samotného řídícího modulu. Program může mít části kódu s chybami, které se ale projeví až při dlouhodobém používání nebo třeba při zvýšené zátěži a kombinaci různých paralelně fungujících větví programu.

Mohou přitom vést například až k vyčerpání paměti zařízení nebo přivedení sběrnice do zakázaného stavu a její zablokování. Bude tedy třeba klást velký důraz na práci s pamětí, kontrolu vstupních dat a zamykání při paralelním přístupu.

4.3.1 Zamykání

Základem je správné zamykání. K dispozici máme přitom výlučné zámky a semafory, a využíváme implementaci Readers-Writers.

Problémy mohou vzniknout na několika místech. Prvním je obecně fakt, že máme paralelně běžící procesy pro obsluhu rozhraní a zároveň sběr dat ze zařízení. Ty se mohou sejít, což by vedlo k poškození obsahu databáze nebo některých nastavení.

S tím pak souvisí i práce s hardwarem. Používáme totiž s I^2C sběrnicí, u které se pro čtení i zápis musí zadávat postupně několik požadavků, které nejprve nastavují adresy a až poté operace. Pokud je nezapíšeme celé nebo zadáme nesprávnou konfiguraci, sběrnice se stane nedostupnou a je třeba zařízení resetovat. To přitom zatím není softwarově možné a pojítko je tak nutné odpojit od zdroje napájení.

Těmto rizikům se musíme vyvarovat. Ani při kladení velkého důrazu na správné zamykání ale nedokážeme jednoduše zajistit, že nedojde k bezvýchodné situaci[11], zároveň také budeme muset podchytit, zda jsme někde na zamykání nezapomněli.

Jediným způsobem, jak tyto chyby detekovat, bude testování chování v různých situacích a zátěži. Jelikož lze modul spouštět i na dalších platformách, můžeme k tomu využít i některých komplexnějších řešení k tomu určených.

Jedním z nich je helgrind[40], který dokáže některé rizikové přístupy do pamětí odhalit. Pokud jej tak použijeme při testování, může nám pomoci odhalit alespoň část případných chyb. Bohužel tak ale neotestujeme správné používání sběrnice, protože v době psaní práce jej na hardwaru nebylo možné spustit.

4.3.2 Správa paměti

Dalším problémem je správa paměti. Příliš velká náročnost na množství alokovaného prostoru a nevhodné uvolňování může vést k jejímu vyčerpání. Budeme tedy muset nějak otestovat, zda k tomu někde nedochází a časem nebude paměť chybět. Situace je ale bohužel podobná, jako u zamykání.

Pro některé platformy na to existují nástroje a můžeme využít například valgrind[41]. Ten umí sledovat přístupy do paměti a umožňuje odhalit špatnou práci s pamětí. Na pojítku jej ale spustit nelze a budeme se muset spolehnout na výsledky průběžného testování.

4.4 Vnější útoky

Dále se zaměříme na vnější útoky, které ohrožují modul přes rozhraní dostupná po síti. Podíváme se na jejich druhy, rizika pro síťová rozhraní a možná řešení.

4.4.1 Varianty

Útoky se budou lišit svým cílem a potřebnými prostředky. V podstatě se můžeme snažit o získání práv superuživatele, kdy pak můžeme se zařízením dělat prakticky cokoliv, nebo alespoň omezit jeho funkčnost a použitelnost zařízení, abychom poškodili zákazníka nebo jeho dodavatele.

[11] problém deadlocku: http://en.wikipedia.org/wiki/Deadlock

4.4.1.1 Získání přístupu

Získání přístupu k zařízení může probíhat více způsoby a mít různá rizika.

Nejjednodušší je získání přístupových údajů. To je problém, protože útočník může pak zařízení libovolně měnit konfiguraci. Dobré ale je, že zákazníkovi to bude způsobovat problémy, časem to odhalí a údaje změní na nějaké bezpečnější nebo zařízení resetuje. Zákazníkovi se to tedy vyplatí řešit, samotné zařízení se přitom nepoškodí a není ohrožena licenční politika.

Rozhraní mohou ale obsahovat chyby, které umožní získání mnohem silnějšího přístupu. Může tak být potom možné spouštět příkazy, které by mohli umožnit zcela přebrat kontrolu nad pojítkem. To by přitom mohlo být v kritickém případě možné i bez správných přihlašovacích údajů a navíc by to umožnilo obejít licenční politiku.

Využít by se mohlo několik cest. Kromě chyby v SSH démonu, který lze použít pro vzdálený správcovský přístup k zařízení, by mohla být chyba v knihovnách pro SNMP nebo HTTP. V kombinaci s chybou v jádře by pak mohl útočník být schopen zvýšit svá práva a získat superuživatele.

Abychom tomu zabránili, bylo třeba maximálně omezit práva procesů, které rozhraní spravují a zároveň aktualizace softwarového vybavení. Zvláštní důraz byl pak kladen v místech, kde bylo třeba spouštět příkazy pro konzoli, aby nemohl útočník snadno spustit vlastní kód. Vlastní firmware obsahuje přitom minimum nástrojů kvůli šetření paměti, máme tedy dobré šance, že běžné útoky selžou na jejich absenci.

Velké riziko pak představují také chyby přetečení zásobníku. Budeme si tedy muset dávat pozor na práci s ukazateli a zpracovávání dat od uživatelů. Mnoho toho za nás ale řeší knihovny a bude tedy hlavně důležité udržovat je aktualizované.

4.4.1.2 Odmítnutí služby

Dalším problémem je odmítnutí služby. Tyto útoky mohou způsobit dočasnou nepoužitelnost jejich cíle. To by mohl být problém, protože bychom nemohli provádět správu zařízení. Útok by zároveň odčerpával část poskytovaného přenosového pásma a měl by tak pravděpodobně negativní vliv na výkon z pohledu zákazníka. Při velké zátěži by se navíc mohla projevit nějaká další chyba a mohlo by to vést na různé pády modulu.

Obecně bude přetížení našeho zařízení poměrně snadné, protože má málo prostředků. Bude tedy dobré zabezpečit síť na vyšší úrovni a sledovat například množství zasílaných požadavků z různých zdrojů a případně jim omezovat přístup. Testováním bychom pak měli podchytit, že ani při zátěži nebude docházet k pádům modulu.

4.4.2 Rozhraní

Z hlediska rozhraní, která mohou být ohrožena, se budeme potýkat prakticky se třemi.

4.4.2.1 SSH

Základní je přístup přes SSH. Ten by měl sloužit pouze pro přístup správců z SVM s.r.o. a je zabezpečen podle práce o bezpečnosti [3].

Důležité bude, aby byl související software aktualizovaný. Toho docílíme pravidelnými aktualizacemi a sestavováním nových verzí firmwaru. Pro vlastní nahrání firmwaru pak bude třeba nastavení již zmíněných firemních procesů.

Druhý problém pak může být v přístupovém heslu k rozhraní. Otázka volby přístupových hesel je na straně SVM s.r.o., bylo by vhodné používat velmi komplexní přístupové údaje, různé pro jednotlivá zařízení, protože jejich prolomení v tuto chvíli znamená možnost přihlásit se jako superuživatel.

4.4.2.2 SNMP

Další je SNMP, zajišťované knihovnou NetSNMP. Její hlavní proces poskytuje rozhraní a náš modul s ním komunikuje prostřednictvím AgentX protokolu.

Knihovna sama o sobě je dlouhodobě vyvíjena a testována. Na nás tedy bude hlavně zajistit případné aktualizace.

Navíc můžeme zvážit její vlastní testování hrubou silou přes vlastní protokol. To může být časově velmi náročné, ale dokáže teoreticky odhalit jakoukoliv chybu. Příkladem nástroje, který se toto snaží implementovat, je snmpfuzzer[42]. Při pokusném testování ale neodhalil žádný problém. Je možné, že testování podobnou formou budou provádět již i samotní autoři.

Z hlediska přístupu je používáno ověřování pomocí jména a hesla. Je přitom možné nastavit přístupy přes SNMP ve verzích 1, 2c i 3. Bezpečnost se pak odvíjí od kvality zvolených přihlašovacích údajů a nastavení přístupu do stromu informací.

Konkrétní přístupy řeší sám vlastník zařízení, je ale důležité, aby kvalitu údajů nepodcenil. Přes SNMP je totiž možné jak zjistit mnoho údajů o nastavení, tak i některé měnit, čímž by mohlo dojít k narušení fungování sítě.

Na úrovni konfigurace SNMP by se pak mohlo objevit riziko v nějaké speciální direktivě[12]. Konfiguraci totiž může administrátor zadat celou a vlastní proces SNMP je přitom spouštěn pod superuživatelem a mohlo by se mu tedy podařit tímto způsobem na zařízení spustit nějaký vlastní kód.

Při zkoumání direktiv by mohl být problém v zadání příkazů pro jazyk Perl. Jeho podpora ale byla odebrána a mělo by tak být maximálně možné zadat špatnou konfiguraci, což ale může pouze zabránit démonu ve správném spuštění.

[12]snmpd.conf: http://www.net-snmp.org/docs/man/snmpd.conf.html

Protokol může pak také představovat riziko pro ostatní uzly v síti. SNMP totiž umožňuje zasílat i poměrně velké odpovědi na dotazy jako GET BULK. Ty pak mohou vytvořit až (D)DoS útok při podvržení zdrojové adresy[43].

Proti tomu se ale nedokážeme jednoduše bránit na úrovni zařízení. Řešením bude dobře zabezpečit přístup, nastavit filtry provozu v síti a zajistit její sledování.

Z pohledu licenční politiky přes SNMP zatím není možné ovlivnit nastavení přenosů. Pokud se ve vlastní knihovně tedy neobjeví závažnější chyba, neměl by pro ni v tomto stavu představovat vážnější riziko.

4.4.2.3 Web

Další cestou by pak mohlo být využití webového rozhraní. Zde je využita používaná knihovna libmicrohttpd a spoléháme na to, že nejzávažnější problémy jsou již tady podchyceny.

Podobně tomu pak je i u knihoven pro SSL nebo nástrojů pro šifrování aktualizačních balíčků. V poslední době se ukázalo, že chyby se mohou vyskytnout i zde. Nedokáže je ale jednoduše podchytit a budeme se spoléhat na aktualizace.

Z hlediska možného útoku na webový server byla snaha alespoň minimalizovat škody. Webový server funguje v samostatném procesu pod jiným uživatelem a má tak omezený přístup do souborového systému i ke zdrojům.

Při teoretickém získání přístupu do prostoru procesu má pak útočník ale k dispozici rozhraní dalšího procesu a případný může pokračovat a zaměřit se na něj. Ten již přitom má práva superuživatele a při úspěchu tak útočník získá úplný přístup. Toto řešení tedy určitě není nedobytné, ale přidává tak alespoň další vrstvu a útok komplikuje.

Autentizace využívá HTTP Basic přes SSL. Pro šifrování se pak generuje sám sebou podepsaný certifikát při prvním pokusu o spuštění HTTP modulu. To bohužel přináší další bezpečnostní problémy.

Základem jsou klíče certifikátu. Nemůžeme mít příliš dlouhé, protože to má velký vliv na rychlost přístupu k rozhraní a generování delších klíčů navíc trvá výrazně déle. Při teoretickém prolomení přitom můžeme získat přístup ke komunikaci a tím i k údajům HTTP Basic autentizace. Použitá délka 2048 bitů by však měla být v této době ještě dostačující a toto riziko by při vhodné konfiguraci SSL nemělo být vysoké.

Jelikož certifikáty nejsou podepisované certifikační autoritou, vystavujeme se Man-in-the-Middle útoku[44]. Certifikát může být na veřejné síti podvrhnut, komunikace bude odposlouchávána a administrátorské heslo vyzrazeno. Neměli bychom k zařízení tedy v takovém případě raději přistupovat.

Z hlediska autentizace opět platí možnost prolomení přihlašovacích údajů. Zde nám trochu nahrává alespoň nižší výkon zařízení, který jej dělá časově velmi náročným. Bude ale hodně záležet na kvalitě nastavených hesel.

Pokus o něj zároveň bohužel může vést na přetížení zařízení. Zabezpečení

bude ale třeba řešit na jiné úrovni v rámci sítě[13], výkonnost tohoto zařízení totiž není ideální pro provoz firewallu a sám o sobě by jej mohl velmi vytěžovat.

K webovému rozhraní patří také přístup přes webový prohlížeč. Zde asi největší problém bude představovat Cross-site scripting[45]. Mohlo by se nám totiž stát, že uživatel zůstane přihlášen k zařízení a například otevře nějaký speciálně upravený odkaz, který nějak upraví jeho nastavení.

Bránit bychom se mohli více způsoby. Mohli bychom například zavést klíč pro požadavek, který se bude postupně měnit a nepůjde znovu použít.

Riziko výskytu by ale mohlo být poměrně malé. Útočník by totiž musel znát adresu zařízení a administrátor by se nesměl odhlásit. Zároveň přitom nepředpokládáme, že do administrace bude přistupovat nějak pravidelně. Proti tomuto útoku se tedy dokážeme poměrně dobře bránit nastavením vhodných procesů pro přístup.

4.5 Vnitřní útoky

Z hlediska vnitřních útoků budou možnosti a rizika podobná, jako u vnějších. Navíc nám ale přibývají útoky fyzické a hlavně problém, že samotný zákazník nemá zájem na tom, aby byly chyby opraveny. Mohlo by se mu totiž například podařit využít některého nedostatku a obejít licenční politiky.

Tato problematika je hodně řešena v rámci procesu aktualizace firmwaru [1]. Je v zájmu SVM s.r.o., aby byli schopni odhalit, že se zařízením bylo manipulováno, musí to být ale v možnostech zařízení.

Rizika se bohužel nepodařilo zcela odstranit. Implementace počítá s šifrováním firmwaru unikátním klíčem zařízení. Po dešifrování má přitom navíc docházet ke kontrole obrazu před nahráním, kde součet je šifrován asymetricky klíčem firmy.

Jak klíč zařízení pro symetrické šifrování, tak i ten pro asymetrické dešifrování musí ale být součástí firmwaru a může být tedy nahrazen. Útočník pak může vytvořit vlastní firmware, upravit data v dočasné paměti zařízení a nahrát jej.

Nepomohla by ani nepřepisovatelná paměť pro klíče. Zkušený útočník by teoreticky mohl přímým přístupem k zařízení získat software, upravit jej tak, aby kontroly neprováděl, a nahrát zpět do paměti zařízení.

Implementovaná opatření tedy riziko neodstraňují. Cestu alespoň komplikují a je na dalším vývoji a testování, aby se neobjevily chyby, které by to zjednodušily.

[13] firewall: http://en.wikipedia.org/wiki/Firewall_(computing)

Kapitola 5

Testování

Závěrečnou kapitolou je testování. To tvoří důležitou součást celého vývoje a snažíme se jím předcházet výskytu co největšího množství chyb. Vytvořené skripty byly přiloženy ke zdrojovým kódům.

5.1 Testy výkonnosti

Začneme výkonností vnějších rozhraní, abychom si mohli udělat lepší představu o použitelnosti. Z hlediska požadavků nebyly stanoveny mezní hodnoty, musí být ale možné provádět sbírání hodnot bez výpadků spojení.

5.1.1 SNMP

Začneme SNMP rozhraním, které je zajímavé hlavně při centralizovaném sledování. Nejsou na něj kladeny vysoké nároky co do množství obsloužených požadavků za čas. Čím rychlejší ale bude, tím kratší dobu bude zatěžovat sledovací server a o to více zařízení pak zvládne kontrolovat.

Test budeme spouštět se základní vytvořenou konfigurací pro pojítko SkyWire. Změříme, kolik času je potřeba na tři průchody tabulkou nástrojem snmpwalk, který opakovaně zasílá dotazy GET NEXT, a kolik dotazů se zařízení za tu dobu podaří obsloužit.

Měření provedeme postupně pro jednoho a dva dotazující se klienty s verzemi 1, 2c a 3. U verzí 1 a 2 použijeme ověření textovým klíčem (community-string), u verze 3 vyzkoušíme šifrování provozu algoritmem DES[14] a hesla zabezpečená otisky MD5[15]. Výsledky měření jsou v tabulce 5.1.

Z testů je vidět, že při paralelizaci klesá počet vyřízených požadavků pro jedno vlákno přibližně na polovinu. Je to dáno možnostmi zařízení, které má pouze jeden procesor s jedním jádrem. Počet obsloužených požadavků je ale i tak víc než dostačující a není třeba jej tedy v tuto chvíli nějak vylepšovat.

[14]DES: http://en.wikipedia.org/wiki/Data_Encryption_Standard
[15]otisk MD5: http://en.wikipedia.org/wiki/MD5

Tabulka 5.1: Výsledky měření výkonnosti SNMP rozhraní

	Požadavků za sekundu	
Verze	1 klient	2 klienti
1	85	47
2c	84	48
3	77	42

5.1.2 Webové rozhraní

Dále se podíváme na rozhraní HTTP(S) a otestujeme několik základních akcí přes SSL s autentizací. Použijeme k tomu nástroj siege[16], který dokáže opakovaně přistupovat na zadanou adresu z definovaného počtu vláken a počítat některé statistické údaje, mezi kterými je i průměrná doba trvání. Měření budeme pro každou kombinaci adresy a počtu vláken provádět deset minut.

Webové rozhraní je určeno pro správu zařízení. Požadavků na něj nebude tedy většinou mnoho a nejsou ani kladeny vysoké nároky na rychlost jejich vyřízení.

Rozhraní ale musí být použitelné. V době kdy proti němu není veden žádný útok, by doba, potřebná na vyřízení požadavku, neměla přesáhnout jednotky sekund a nemělo by docházet ani k jeho náhlé nedostupnosti.

Testované požadavky byly:

- Informace - Stažení informací o položkách. Jde o poměrně velké množství dat, která se ale nemusejí stahovat opakovaně.
- Struktura - Struktura položek. Opět spíš statický obsah, který je navíc menší co do objemu dat, ale náročnější na vygenerování.
- Hodnoty - Stažení hodnot položek ve skupině. Menší množství dat, kde je ale náročnější získání samotných aktuálních hodnot při generování.

Tabulka 5.2: Výsledky měření výkonnosti webového rozhraní

Funkce	1	2	3	4	5	6	7	8	9	10
Informace	0.43	0.59	0.66	0.69	0.71	0.72	0.73	0.74	0.74	0.74
Struktura	0.40	0.56	0.64	0.68	0.71	0.70	0.71	0.72	0.72	0.72
Hodnoty	0.35	0.53	0.60	0.66	0.69	0.68	0.71	0.71	0.72	0.72

Rychlost bohužel není nijak vysoká. Využití procesoru se mírně zlepší při paralelním přístupu, i tak ale budeme často potřebovat na vyřízení jednoho

[16]nástroj siege: http://www.joedog.org/siege-home/

požadavku běžně okolo dvou sekund. Při použití klasického webového prohlížeče budou ale přístupy rychlejší, protože se budou zachovávat spojení, díky zasílání HTTP hlavičky Keep-Alive[17].

Vzhledem k určení pro správu je to ale i tak dostatečné. V praxi předpokládáme, že konfigurace by mohla být měněna pomocí skriptů a bude tedy možné provádět nastavení pro více zařízení z více procesů najednou a správa by tak mohla být výrazně rychlejší.

5.2 Funkční testy

Z hlediska spolehlivosti bude důležité správné chování jednotlivých funkcí. Byl tedy připraven skript na jejich otestování proti spuštěnému zařízení.

5.2.1 SNMP

Začneme se SNMP, kde lze očekávat, že vlastní knihovna je funkční. Důležité tedy bude správné zpracování dotazů na straně modulu.

Základní byla možnost upravovat mu konfiguraci. Pomocí webového rozhraní tedy je zkoušena změna konfigurace přístupových práv a možnost dotazovat se na vybrané OID. Očekávána je přitom základní konfigurace pro SkyWire a při jiném nasazení bude test pravděpodobně třeba upravit.

Příliš velká konfigurace by mohla vyčerpat paměť na zařízení. Implementován byl tedy také limit na deset kilobajtů a testujeme, zda pokus o nahrání větší selže.

Špatná konfigurace může vést na nefunkčnost SNMP rozhraní. Jelikož ale administraci není žádoucí omezovat ovládacím rozhraním a vytvořením rozhraní nad každou konfigurační direktivou SNMP, správné vyplnění bude i nadále na administrátorovi a z hlediska řídícího modulu musí být modul schopen konfiguraci i při chybě nahradit a provést restart hlavního procesu SNMP. Na to byl tedy také doplněn test.

Kromě dotazů na data umožňuje SNMP také hodnoty měnit a před produkčním nasazením bude vhodné připravit pro každou zapisovatelnou datovou položku specifické testy. Na ukázku byla zvolena jedna a připraven test zápisu a vyčtení hodnoty, což prokazuje, že ze SNMP do paměti vede funkční čtecí i zapisovací cesta.

5.2.2 Web

Testovat bylo třeba i webové rozhraní, jehož funkce budou po nasazení pro administrátora prakticky jedinou cestou, jak vzdáleně ovlivnit konfiguraci.

[17] HTTP Keep-Alive: http://en.wikipedia.org/wiki/HTTP_persistent_connection

5.2.2.1 Rozhraní

Základem jsou funkce pro správu rozhraní a restart zařízení. Byly tedy připraveny základní testy pro změnu přístupových údajů a možnost jejich znovupoužití po restartu zařízení a tím máme zároveň ověřeno, že funguje zápis konfigurace na flash.

Při restartu bylo pouze třeba zohlednit čekání na start zařízení a test byl doplněn o uspání. Pro urychlení by mohlo být vhodné doplnit kratší čekání a opakované pokusy. Je zde zároveň možné později doplnit i ověření, zda bude zařízení schopné se stát plně funkčním do nějaké určité doby.

5.2.2.2 Firmware

Další důležitá volání souvisela s aktualizací firmwaru. Zde byl tedy implementován test celého navrženého procesu a v rámci něj se provádí jak stažení a ověření balíčku, tak i vygenerování fiktivního nového a pokus o jeho nahrání.

Využívá se přitom testovacích klíčů a toho, že modul vlastní přehrání neprovádí. Po dokončení implementace tohoto procesu na nižší úrovni bude nutné test upravit a nejspíš bude vhodné zavést parametr, který přehrání při testování zamezí, protože by testování výrazně zpomaloval.

Z hlediska bezpečnosti zde bylo objeveno riziko při vícenásobném nahrávání, které by mohlo vést k poškození balíčku na straně zařízení. To za nás ale automaticky řeší proces ověření, který selže na kontrole obsahu a balíček smaže.

Druhé riziko je v jeho velikosti, kdy při nahrávání by mohla dojít paměť v zařízení, a mohly by se začít ukončovat procesy. V modulu byla tedy omezena jeho maximální velikost a očekáváme, že pokud po nahrání firmwaru nebude paměť stačit, dojde k pádům procesů a některé další testy selžou.

Při praktickém testování k výpadkům nedocházelo, později ale může být třeba tento proces upravit kvůli dalším rozšířením a změnám nároků. V úvahu by pak pro ušetření paměti připadalo dočasné ukončování některých procesů, jako je SNMP, nebo alokace dalších sektorů ve flash paměti a jejich využití jako dočasného úložiště před vlastním úplným přepisem.

5.2.2.3 Data

Kromě poskytnutí možností pro správu je pak u řídícího modulu asi nejdůležitější sběr a poskytování informací ze zařízení. K tomu je k dispozici řada volání, jejichž funkčnost byla opět otestována na možnost úspěšného získání odpovědi.

V úvahu pak připadá další vylepšení ověřování struktury zpráv, až bude upevněna. K tomu ale dojde až po dopsání webového rozhraní a v tuto chvíli se dá očekávat, že ještě bude docházet ke změnám. Zatím to tedy nebylo implementováno.

5.2.2.4 Sledování

S daty pak souvisí jejich sledování. Zde byla klíčová možnost spravovat kontroly a jejich operace. Byla tedy vytvořena sada testů, které operace zkouší na různé případy.

Důležité je hlavně správné sestavení kontrolního výrazu. Jeho kontrola může být při špatné definici poměrně náročná a zabrat nezanedbatelný čas, který prodlouží intervaly mezi sbíráním dat.

Kvůli riziku nedostatku místa pro uživatelskou paměť je toto alespoň částečně technicky sníženo omezením délky výrazu a jejich množství. Může se ale ukázat, že bude třeba jej navýšit a pak je třeba mít toto na paměti.

5.2.2.5 Zařízení

Důležitá je pak také možnost nastavovat parametry periferií, kde máme v tuto chvíli Ethernetový port. Byla tedy napsána sada testů pro postupné změny jeho nastavení a kontroly provedení.

Bylo přitom důležité ověřit, zda budou do konfigurace uloženy pouze použitelné hodnoty. Testují se tedy i špatné vstupy a ověřujeme tak, zda na straně modulu jsou tato případná rizika narušení konfigurace zohledněna.

Nastavení přitom probíhá často voláním externích programů. Je tedy také důležité vstupy správně kontrolovat a filtrovat, aby nemohl být spuštěn nějaký další příkaz a nemohla být ohrožena bezpečnost zařízení.

Uložení špatného vstupu by přitom mohlo vést na znepřístupnění po síťovém rozhraní. Bylo by pak nutné opět nahrát firmware přes lokální rozhraní, což by bylo pro zákazníka bylo velmi komplikované.

5.2.2.6 Soubory

Rozhraní musí také být schopno poskytovat soubory uživatelského webového rozhraní ze souborového systému. Jeho fungování testujeme základními pokusy o stažení jeho hlavního souboru „index.html“.

Přístup by měl být kromě přístupu k HTTP rozhraní modulu omezen pouze na soubory z jedné pevně dané složky. Testy jsou tedy také doplněny o pokus o stažení standardního souboru „/etc/passwd“ na OS Linux pomocí úprav adresy a měl by přitom selhat.

Obrana proti těmto pokusům může být poměrně složitá. Existuje ale pomocná funkce „realpath“[34], která převádí zadanou cestu na skutečnou a modul porovnává cestu proti přeložené. Z testů to vypadá, že je použita správně a přístup k ostatním souborům mimo složku není jednoduše možný.

5.3 Chování v zátěži

To, že bude modul fungovat správně při nízké zátěži, ještě nemusí znamenat, že tomu tak bude i při nějaké vyšší. Potřebujeme, aby byl modul dlouhodobě

stabilní a nedocházelo k jeho pádům ani při delším náročnějším používání nebo i přímo po pokusech o narušení jeho funkčnosti v rámci nějakého útoku a i pro tyto případy byl tedy testován.

5.3.1 Zátěž

Základem bylo zvážit, co pro nás zátěž bude. Dá se očekávat, že zařízení bude dlouhodobě monitorováno přes SNMP rozhraní, což lze simulovat soustavným dotazováním na data. K tomu můžeme použít podobný postup, jako u již existujícího řešení pro testování výkonnosti SNMP rozhraní s tím, že dotazování necháme běžet po delší dobu.

Dále budeme provádět různé administrativní úkony přes webové rozhraní. V praxi budou probíhat méně často, pro naše potřeby urychlení je ale můžeme provádět soustavně a opět po delší dobu.

U obojího bychom přitom potřebovali otestovat, zda nebudou vznikat problémy při paralelním prováděním různých činností. Vše tedy necháme probíhat dohromady.

5.3.2 Realizace

V rámci realizace vznikl skript, který tyto procesy spouští. Udržuje několik paralelně spuštěných procesů, které zároveň provádějí průchody SNMP tabulkou a některé z požadavků na webové rozhraní, čímž vytváří soustavnou zátěž.

Pokud tedy budeme mít někde problémy s úniky paměti nebo například nesprávně zamykat, máme docela dobré šance tato místa odhalit. Ani při několika hodinovém provozu se ale nevyskytly problémy se stabilitou a vypadá to, že modul by mohl být použitelný.

5.4 Shrnutí

Rozsáhlým testováním dokážeme pokrýt mnoho možných problémů. Hodně se jich v průběhu testování ukázalo a byly opraveny.

Nikdy ale nedokážeme postihnout úplně vše. Testy bude nutné udržovat a doplňovat. Pravidelné dlouhodobé testování jak při vývoji, tak i po aktualizaci balíčků firmwaru bude jediný způsob, jak alespoň částečně omezit výskyt základních chyb.

Závěr

V průběhu tvorby práce se ukázalo, že splnění zadaných požadavků bude výrazně náročnější, než se původně předpokládalo. Rozšíření si vyžádalo zásahy, které nebyly vůbec v plánu.

Práce přitom vyžadovala výrazné doplnění znalostí. Bylo nutné se seznámit jak s prostředím, tak i s pojítkem, firmwarem a jeho možnostmi.

To samé platí i ve spojení s použitým hardwarem, kde bylo nutné uvažovat i přístup k zařízení na nižší úrovni. Hlavně použití I^2C sběrnice, spojené s hledáním řešení v existujícím, ale ne vždy funkčním, kódu, se ukázalo jako velmi náročné.

Problémy byly také s výkonem a dostupnou pamětí. Na tato kritéria bylo po celou dobu nutné nahlížet a náležitě upravovat návrh a vytvářené řešení.

Vše doplňovaly problémy s kompatibilitou kódu a nečekanými pády po celou dobu tvorby. Jejich ladění bylo často náročné, protože docházelo k odlišnostem v chování v testovacím prostředí a na pojítku.

Zadání se ale i přesto podařilo splnit. Byla postupně provedena analýza a vytvořeno rozšiřitelné řešení, které kromě plnění původních funkcí podporuje také SNMP V3, je schopno nejen sledovat hodnoty parametrů zařízení, ale i udržovat jejich historii, a řeší také aplikační část webového rozhraní a přidává podporu pro proces aktualizace firmwaru samotného zařízení.

Výsledný modul byl pak integrován do sestavovacího prostředí a upravený firmware testován přímo na pojítku. Po celou dobu byl přitom kladen velký důraz na co nejnižší nároky na prostředky v kombinaci s vysokou bezpečností a výsledky byly rovněž doplněny do této práce.

Na implementaci je stále velký prostor pro další rozšiřování a zlepšování. Všechny funkční i nefunkční požadavky na konfiguraci, data, SNMP, webové rozhraní i na aktualizace byly pro teď v rámci možností nicméně splněny.

Práci je nyní možné přímo použít na dodaném zařízení a vše je připraveno pro dokončení a nasazení v produkčním prostředí. Výsledné řešení je přitom poměrně obecné a s menšími úpravami jej bude možné použít i na dalších zařízeních s OS Linux.

Literatura

[1] Šrámek, P.: *Pojítko SkyWire - Ovladač paměti flash.* 2013. Dostupné z: https://dip.felk.cvut.cz/browse/pdfcache/sramepa1_2013dipl.pdf

[2] Sandr, O.: *Pojítko SkyWire - Moduly pro aktualizaci zařízení.* 2013. Dostupné z: https://dip.felk.cvut.cz/browse/pdfcache/sandrota_2013dipl.pdf

[3] Šmarda, M.: *Pojítko SkyWire - Zabezpečení a ochrana před neoprávněnou manipulací.* 2013. Dostupné z: https://dip.felk.cvut.cz/browse/pdfcache/smardmar_2013dipl.pdf

[4] Ellingwood, J.: *An Introduction to SNMP (Simple Network Management Protocol).* 2014, [cit. 2014-11-17]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-snmp-simple-network-management-protocol

[5] Leskiw, A.: *SNMP Tutorial Part 2: Rounding Out the Basics.* [cit. 2014-11-16]. Dostupné z: http://www.networkmanagementsoftware.com/snmp-tutorial-part-2-rounding-out-the-basics

[6] *Tembria SNMP Browser tm - Free!* [cit. 2014-11-15]. Dostupné z: http://www.tembria.com/products/snmpbrowser/

[7] *Introducing JSON.* [cit. 2014-11-17]. Dostupné z: http://json.org/

[8] *Extensible Markup Language (XML).* [cit. 2014-11-16]. Dostupné z: http://xml.coverpages.org/xml.html

[9] *XML Schema Part 0: Primer Second Edition.* [cit. 2014-11-16]. Dostupné z: http://www.w3.org/TR/xmlschema-0/

[10] *When to Use SAX.* [ciz. 2014-10-15]. Dostupné z: http://docs.oracle.com/javase/tutorial/jaxp/sax/when.html

[11] *CGI: Common Gateway Interface*. [cit. 2014-10-15]. Dostupné z: http://www.w3.org/CGI/

[12] Shenoy, P.: *Synchronization for Readers/Writers Problem*. [cit. 2014-10-15]. Dostupné z: http://lass.cs.umass.edu/~shenoy/courses/fall08/lectures/Lec11.pdf

[13] *Buildroot*. [cit. 2014-10-15]. Dostupné z: http://buildroot.uclibc.org/about.html

[14] *NetSNMP*. [cit. 2014-09-20]. Dostupné z: http://www.net-snmp.org/

[15] *Socket programming and the C BSD API*. [cit. 2014-11-17]. Dostupné z: http://www.yolinux.com/TUTORIALS/Sockets.html

[16] Barney, B.: *POSIX Threads Programming*. 2014, [cit. 2014-11-17]. Dostupné z: https://computing.llnl.gov/tutorials/pthreads/

[17] *BusyBox*. [cit. 2014-11-16]. Dostupné z: http://www.busybox.net/

[18] *SH7760*. [cit. 2014-09-16]. Dostupné z: http://www.renesas.com/products/mpumcu/superh/sh7750/sh7760/index.jsp

[19] Shukla, V.: *Linux threading models compared: LinuxThreads and NPTL*. [cit. 2014-09-16]. Dostupné z: http://comet.lehman.cuny.edu/jung/cmp426697/NPTL.pdf

[20] *The Net-SNMP Wiki*. 2014, [cit. 2014-11-17]. Dostupné z: http://www.net-snmp.org/wiki/index.php

[21] *An Introduction to the Autotools*. [cit. 2014-11-16]. Dostupné z: http://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html

[22] *PugiXML*. [cit. 2014-09-21]. Dostupné z: http://pugixml.org/

[23] erik: *Set Interface IP Address From C In Linux*. 2014, [cit. 2014-11-17]. Dostupné z: http://www.lainoox.com/set-ip-address-c-linux/

[24] *ifconfig(8) - Linux man page*. [cit. 2014-11-17]. Dostupné z: http://linux.die.net/man/8/ifconfig

[25] Amos, B.: *CPP Expression Parser*. [cit. 2014-09-23]. Dostupné z: https://github.com/bamos/cpp-expression-parser

[26] Chorafakis, D.: *Building a Net-SNMP subagent*. [cit. 2014-09-28]. Dostupné z: http://www.chorafakis.com/2011/12/20/building-a-net-snmp-subagent/

[27] *Writing a Subagent*. [cit. 2014-09-28]. Dostupné z: http://www.net-snmp.org/wiki/index.php?title=TUT:Writing_a_Subagent&oldid=5770

[28] Bernat, V.: *Asynchronicity & Net-SNMP AgentX protocol*. [cit. 2014-09-28]. Dostupné z: http://vincent.bernat.im/en/blog/2012-fixing-async-agentx.html

[29] Mauro, D. R.; Schmidt, K. J.: SNMP Operations. In *Essential SNMP*, 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., druhé vydání, 2005, ISBN 0-596-00840-6, s. 37–63, [cit. 2014-09-28].

[30] *GNU Libmicrohttpd*. [cit. 2014-09-28]. Dostupné z: http://www.gnu.org/software/libmicrohttpd/

[31] *A tutorial for GNU libmicrohttpd*. [cit. 2014-09-28]. Dostupné z: http://www.gnu.org/software/libmicrohttpd/tutorial.html

[32] Barker, E.; Roginsky, A.: *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. [cit. 2014-09-28]. Dostupné z: http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf

[33] Jiang, H.: *JSON++*. [cit. 2014-09-20]. Dostupné z: https://github.com/hjiang/jsonxx

[34] *realpath*. [cit. 2014-09-29]. Dostupné z: http://pubs.opengroup.org/onlinepubs/9699919799/functions/realpath.html

[35] *signal(7) - Linux man page*. [cit. 2014-10-16]. Dostupné z: http://linux.die.net/man/7/signal

[36] Podhola, M.: *Signály a procesy 2*. 2005. Dostupné z: http://www.linuxexpres.cz/praxe/signaly-a-procesy-2

[37] *Shellshock Bug: 6 Key Facts*. [cit. 2014-10-01]. Dostupné z: http://www.informationweek.com/government/cybersecurity/shellshock-bug-6-key-facts--/d/d-id/1316131

[38] codenomicon: *The Heartbleed Bug*. [cit. 2014-10-01]. Dostupné z: http://heartbleed.com/

[39] Srinivas: *Practical Shellshock Exploitation – Part 2*. Infosec Institute, [cit. 2015-01-05]. Dostupné z: http://resources.infosecinstitute.com/practical-shellshock-exploitation-part-2/

[40] *Helgrind: a thread error detector*. [cit. 2014-11-16]. Dostupné z: http://valgrind.org/docs/manual/hg-manual.html

[41] *Valgrind*. [cit. 2014-11-16]. Dostupné z: http://valgrind.org/

[42] Orrey, K.: *snmp-fuzzer*. 2008, [cit. 2014-11-16]. Dostupné z: http://www.vulnerabilityassessment.co.uk/snmpfuzzer.htm

[43] Broadband Internet Technical Advisory Group: *SNMP Reflected Amplification DDoS Attack Mitigation*. 8 2012, [cit. 2014-10-13]. Dostupné z: http://www.bitag.org/documents/SNMP-Reflected-Amplification-DDoS-Attack-Mitigation.pdf

[44] *Man-in-the-middle attack*. 2014, [cit. 2014-11-17]. Dostupné z: https://www.owasp.org/index.php?title=Man-in-the-middle_attack&oldid=179899

[45] *Cross-site Scripting (XSS)*. 2014, [cit. 2014-11-17]. Dostupné z: https://www.owasp.org/index.php?title=Cross-site_Scripting_(XSS)&oldid=173274

Příloha **A**

Seznam použitých zkratek

CGI Common Gateway Interface

CSV Comma-separated Value

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

NPTL Native POSIX Threading Library

RAM Random Access Memory

SNMP Simple Network Management Protocol

XML Extensible Markup Language

SAX Simple API for XML

DOM Document Object Model

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS

Příloha C

Dokumentace webového rozhraní

V této příloze jsou rozepsány jednotlivé implementované akce rozhraní. Každá se skládá z HTTP metody a adresy.

U požadavků POST je doplněn tvar zasílaných dat, kde jako formát byl většinou použit JSON. Na straně webového prohlížeče se předpokládá využití jazyka JavaScript.

Navíc je zde ukázkový tvar odpovědí, které jsou opět většinou ve formátu JSON. Odpověď je v tomto tvaru, pokud je HTTP kód odpovědi 200.

Při chybách může rozhraní vracet další kódy. Použití špatných přihlašovacích údajů vede na kód 401. Požadavek na neexistující akci (kombinaci metody a adresy) pak vede na kód 404.

Odpověď úspěšných akcí má standardní tvar, kde je indikována úspěšnost a zpráva. V následujícím výpise akcí je tomu tak všude, kde není uvedeno jinak.

```
{
    "success": true,
    "message": "..."
}
```

C.1 Firmware

C.1.1 GET /api/firmware/update

První krok procesu aktualizace firmwaru. Umožňuje stáhnout balíček pro získání aktualizace firmwaru.

C.1.2 POST /api/firmware/update

Umožňuje nahrát aktualizační balíček. Ten je třeba posílat POST požadavkem v proměnné „file“. Maximální velikost byla stanovena na 7 MB.

C.2 Základní rozhraní

C.2.1 GET /api/info

Základní testovací požadavek. Umožňuje vyzkoušet funkčnost webového rozhraní a do budoucna může poskytovat nějaké další základní údaje.

C.2.1.1 Odpověď

```
{
    "message": "Hello"
}
```

C.2.2 POST /api/info

Požadavek na aktualizaci konfigurace části modulu, poskytující funkce webového rozhraní.

C.2.2.1 Požadavek

Součástí požadavku může být nové heslo, přihlašovací jméno a případně lze změnit i realm HTTP Basic autentizace.

```
{
    "username": "admin",
    "password": "123456",
    "realm": "StationManager"
}
```

C.2.3 POST /api/reboot

Vyvolá restart zařízení.

C.3 Sledování dat

C.3.1 GET /api/data/checks

Získání aktuálně nastavených kontrol.

U akce SNMP trap indikuje vtype typ zasílané informace a value je pak samotnou hodnotou. ID odpovídá OID v případné tabulce SNMP trapů.

C.3.1.1 Odpověď


```
{
  "data": {
    "childs": [{
      "check": [{
        "_check": "(($1.8.1.0$ + $1.8.1.1$) / 2) > 50",
        "_id": "1",
        "childs": [{
          "operation": [{
            "_auth": "1",
            "_id": ".1.2.3",
            "_type": "SNMP_TRAP",
            "_value": "abc",
            "_vtype": "s"
          }]
        }]
      }]
    }]
  },
  "success": true
}
```


C.3.2 POST /api/data/checks

Vytváří novou kontrolu. Kromě samotné kontroly a zvoleného ID (musí vybrat unikátní) je třeba zadat také operace, které se mají stát. Aktuálně je k dispozici pouze SNMP trap, doplněný o identifikátor přihlašovacích údajů. Ty se zadávají samostatně a je pro ně samostatná sada požadavků pro správu dále.

C.3.2.1 Požadavek


```
{
  "check": "",
  "id": "",
  "operations": [{
    "auth": "1",
    "id": ".1.2.3",
    "type": "SNMP_TRAP",
    "value": "abc",
    "vtype": "s"
  }]
}
```

C.3.3 POST /api/data/checks/id

Požadavek na aktualizaci kontroly. Má stejnou podobu, jako ten pro její vytvoření.

C.3.3.1 Požadavek

```
{
  "check": "",
  "id": "",
  "operations": [{
    "auth": "1", "id": ".1.2.3",
    "type": "SNMP_TRAP",
    "value": "abc", "vtype": "s"
  }]
}
```

C.3.4 DELETE /api/data/checks/id

Maže zadanou kontrolu podle ID.

C.4 Přístup k datům

C.4.1 GET /api/data/structure

Vrací strukturu položek. Přes ID skupin a položek se lze dále dotazovat dalšími požadavky.

C.4.1.1 Odpověď

```
{
  "structure": [{
    "items":{},
    "groups":{
      "1":{
        "items": [{ "id": 10001, "info": 1 }],
        "groups": []
      },
      "2":{
        "items":[{ "id": 10002, "info": 2 }],
        "groups":[]
      }
    }
  }]
}
```

C.4.2 GET /api/data/info

Doplňující informace o položkách. Získávají se převodem informací z hlavního konfiguračního XML souboru.

C.4.2.1 Odpověď

```
{
  "data": [{
    "_div": "1000",
    "_id": "1",
    "_max": "2.4",
    "_min": "100.0",
    "_mul": "8",
    "_type": "FLOAT",
    "_unit": "V",
    "_label_en": "Voltage 1.2 V",
    "_label_cs": "Napeti 1.2 V"
  }]
}
```

C.4.3 GET /api/data/id

Požadavek na získání všech nasbíraných dat k položce podle ID.

C.4.3.1 Odpověď

```
{
  "depth": 2,
  "success": true,
  "width": 5,
  "data": [
    [1,1,1,1,1],
    [2,2,2,2,2]
  ]
}
```

C.4.4 POST /api/data/id

Požadavek na nastavení hodnoty položky podle ID. Typ i zdroj to musí podporovat, jinak skončí s chybou.

C.4.4.1 Požadavek

```
{ "data": "1.2" }
```

C.4.5 GET /api/data/group/id/values

Požadavek na vytažení dat ze skupiny podle ID. Na podskupinu se lze dotázat oddělením jejich ID tečkami.

C.4.5.1 Odpověď

```
{
    "success": true,
    "data": [
        {"id":10001,"val":1.216}
    ]
}
```

C.5 SNMP

C.5.1 GET /api/snmp/auths

Zadané autentizační údaje SNMP uživatelů. Důležité pro zasílání SNMP trap.

C.5.1.1 Odpověď

```
{
  "success": true,
  "data": [{
    "_auth": "MD5",
    "_authPass": "123456",
    "_enc": "DES",
    "_encPass": "123456",
    "_engine": "0x8000000001020304",
    "_host": "192.168.1.179",
    "_id": "1",
    "_level": "authPriv",
    "_user": "user",
    "_version": "3"
  }]
}
```

C.5.2 POST /api/snmp/auths

Požadavek na zadání SNMP autentizačních údajů. Použití jednotlivých položek závisí na vybrané verzi, toto je celá struktura požadavku.

C.5.2.1 Požadavek

```
{
  "auth": "MD5",
  "authPass": "123456",
  "enc": "DES",
  "encPass": "123456",
  "engine": "0x8000000001020304",
  "host": "192.168.1.179",
  "id": "1",
  "level": "authPriv",
  "user": "user",
  "version": "3"
}
```

C.5.3 POST /api/snmp/auths/id

Požadavek na editaci SNMP údajů podle ID.

C.5.3.1 Požadavek

```
{
    "id": "5"
    "auth": "MD5",
    "authPass": "123456",
    "enc": "AES",
    "encPass": "123456",
    "engine": "0x8000000001020304",
    "host": "192.168.1.179",
    "level": "authPriv",
    "user": "userx",
    "version": "3"
}
```

C.5.4 DELETE /api/snmp/auths/id

Smazání údajů podle ID.

C.5.5 GET /api/snmp/conf

Získání aktuální specifické části konfigurace SNMP.

C.5.5.1 Odpověď

```
{ "success": true, "data": "rocommunity public\n" }
```

C.5.6 POST /api/snmp/conf

Přepisuje stávající SNMP konfiguraci. Zároveň dojde k restartu SNMP procesu a konfigurace lze tak hned i používat.

C.5.6.1 Požadavek

```
{
    "data": "rocommunity private\n",
}
```

C.6 Zařízení

C.6.1 GET /api/devices

Získání seznamu zařízení a jejich nastavení.

C.6.1.1 Odpověď

```
{
  "success": true,
  "data": [{
    "_id": "1",
    "_ip": "192.168.1.51",
    "_mac": "",
    "_netmask": "255.255.255.0"
  }]
}
```

C.6.2 POST /api/devices/id

Požadavek na aktualizaci zařízení. Dostupné parametry závisí na typu zařízení, v příkladu aktualizujeme nastavení rozhraní Ethernet pro IPv4.

C.6.2.1 Požadavek

```
{
  "id": "1",
  "ip": "192.168.1.52",
  "mac": "74:E5:0B:46:29:15",
  "netmask": "255.255.255.0"
}
```