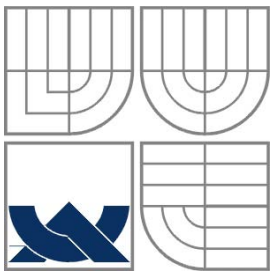
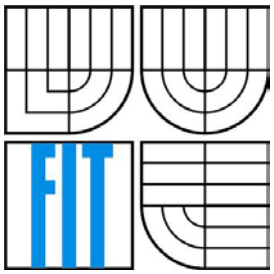


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VZDÁLENÁ SPRÁVA UNIXOVÉHO SMĚROVAČE POMOCÍ PROTOKOLU XMPP/JABBER

REMOTE MANAGEMENT OF UNIX ROUTER WITH XMPP/JABBER PROTOCOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ BEZDÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ KAŠPÁREK

BRNO 2011

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací modulárního klienta protokolu XMPP/Jabber pro interpretaci příkazů určeného pro operační systém OpenWrt.

Práce obsahuje informace o protokolu XMPP/Jabber a operačním systému OpenWrt a popisuje návrh a implementaci klienta.

Abstract

This thesis concerns designing and implementing a modular client for XMPP/Jabber protocol intended for interpreting commands for the OpenWrt operating system.

My work contains information about XMPP/Jabber protocol and OpenWrt operation system and describes the process of designing and implementing the client.

Klíčová slova

Vzdálená správa, Jabber, XMPP, OpenWrt, Iksemel, směrovač

Keywords

Remote management, Jabber, XMPP, OpenWrt, Iksemel, router

Citace

Bezdíček Lukáš: Vzdálená správa unixového směrovače pomocí protokolu XMPP/Jabber, bakalářská práce, Brno, FIT VUT v Brně, 2011

Vzdálená správa unixového směrovače pomocí protokolu XMPP/Jabber

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Tomáše Kašpárka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Bezdíček
18. 5. 2011

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Tomáši Kašpárkovi za odbornou pomoc a další cenné rady, které mi poskytl při zpracování mé bakalářské práce.

Obsah

1 Úvod

Tento dokument popisuje přípravu na řešení a řešení bakalářské práce na téma "Vzdálená správa unixového směrovače pomocí protokolu XMPP/Jabber". Mým úkolem bylo nastudovat z oficiálních dokumentací protokol XMPP/Jabber, seznámit se s operačním systémem OpenWrt a navrhnout klienta protokolu XMPP/Jabber pro interpretaci příkazů a sběr informací o síťovém provozu.

V první části dokumentu je uvedeno seznámení s cílem práce. V druhé části je uveden přehled použitých technologií a jejich popis. V třetí části dokumentu je popsán návrh aplikace následovaný částí zabývající se popisem implementace aplikace a jsou zde uvedeny nároky a vliv výsledné aplikace na cílové zařízení. Na závěr dokumentu je uvedeno shrnutí mých výsledků, kterých jsem dosáhl.

2 O projektu

Cílem této práce je vytvořit aplikaci pro vzdálenou správu směrovačů a monitorování jejich stavu, který bude ke komunikaci využívat protokol XMPP/Jabber.

Program bude interpretovat příchozí textové zprávy, zaslané důvěryhodnými subjekty, jako příkazy a provádět patřičné akce. Dále bude monitorovat stav zařízení a případně informovat uživatele o důležitých událostech (chyby, varování, informační zprávy) v reálném čase prostřednictvím zpráv nebo prezence (stavu). V neposlední řadě bude aplikace umožňovat obousměrný přenos souborů (logy, konfigurační soubory, skripty…), například za účelem zálohování nebo konfigurace.

Aplikace bude vyvíjena pro platformu OpenWrt s ohledem na požadavek na nízké hardwarové nároky, který vyplývá z faktu, že se jedná o aplikaci pro vestavěná zařízení s omezenou výpočetní kapacitou. Psána bude v jazyce C s využitím knihovny iksemel pro implementaci komunikace protokolem XMPP, která byla pro takovéto účely vytvořena. Nadále se bude využívat balík nástrojů OpenWrt Buildroot, který obsahuje veškeré potřebné nástroje pro překlad aplikace pro různé architektury.

Vývoj bude probíhat iterativně a po každé iteraci bude k dispozici spustitelný program s určitou funkčností, kterou bude možno testovat. Testovat se budou nejdříve jednotlivé komponenty a poté se bude testovat projekt jako celek. Testování bude probíhat z počátku na virtuálním stroji s operačním systémem OpenWrt a poté přímo na určitém zařízení. Jako zástupce směrovačů určených do domácností a kanceláří byl zvolen Netgear WNDR3700v2 disponující procesorem o frekvenci 680Mhz, operační pamětí RAM o velikosti 64MB a pamětí Flash o velikosti 16MB. Na tomto směrovači byl původní firmware nahrazen operačním systémem OpenWrt 10.03.1-rc4 s označením „Backfire“.

Součástí práce je i testování vlivu aplikace na výkon cílového zařízení a jeho výsledky budou uvedeny v rámci dokumentu.

3 Použité technologie

V této kapitole jsou uvedeny technologie a prostředky použité při návrhu a implementaci aplikace.

3.1 XMPP

Extensible Messaging and Presence Protocol (XMPP) je otevřený standard pro komunikaci v reálném čase používající Extensible Markup Language (XML) jako základní formát pro výměnu informací. XMPP v podstatě umožňuje posílat malé části XML z jedné entity do druhé v téměř reálném čase.

Jabber/XMPP projekt založil Jeremie Miller v roce 1998 a vytvořil open source server s názvem *jabberd*, který vydal 4. ledna 1999. S pomocí široké komunity vývojářů byli následně vytvořeni klienti pro Linux, Macintosh i Windows; různá rozšíření pro server a knihovny pro jazyky jako jsou například Perl a Java. Během roku 1999 a začátku roku 2000 tato komunita spolupracovala na detailech protokolu nyní nazývaného XMPP. Tato práce vyvrcholila vydáním *jabberd 1.0* v květnu roku 2000.[1]

Účelem XMPP protokolu je umožnit výměnu relativně malých částí strukturovaných dat nazývaných „XML sloky“ přes síť mezi dvěma nebo více entitami.

3.1.1 Architektura

Ačkoli XMPP není spjatý s žádnou konkrétní síťovou architekturou, je postaven obvykle na decentralizované architektuře klient-server, kde se klienti potřebují připojit k serveru, aby získali přístup k síti a umožnili tak výměnu XML slok mezi ostatními entitami. Klienti využívají k přístupu na server TCP protokol a taktéž servery mezi sebou komunikují pomocí TCP protokolu.[2]

3.1.1.1 Server

Server se chová jako inteligentní abstraktní vrstva pro XMPP komunikaci. Primární povinností serveru je spravovat komunikační spojení mezi ním a jinými entitami ve formě XML toků a směrovat patřičně adresované XML sloky mezi těmito entitami.

Většina XMPP serverů dále převzala odpovědnost za uchování dat používané klienty (například seznam kontaktů používaný v IM aplikacích).

Server se může volitelně připojit k jinému serveru, aby umožnil mezidoménovou nebo meziserverovou komunikaci. K tomu je zapotřebí, aby oba servery mezi sebou ustavily spojení k umožnění vzájemnému přenosu XML slok.[2]

3.1.1.2 Klient

Většina klientů se připojuje k serveru přímo protokolem TCP a pomocí XMPP využívá veškerou funkčnost poskytovanou XMPP serverem a přidružených služeb. Více prostředků se může současně připojit k serveru pomocí jednoho účtu, adresy jednotlivých prostředků se pak liší v identifikátoru prostředku (např.: node@domain/home vs. node@domain/work), jak je definováno v adresovém schématu. Doporučené číslo portu pro připojení klienta k serveru je 5222, jak je registrováno organizací IANA.[2]

3.1.2 Základní prvky datového přenosu

Datový přenos mezi dvěma entitami probíhá pomocí persistentních XML datových toků, vytvořených na navázaném TCP spojení. Přes tento XML tok si entity navzájem posílají fragmenty XML zvané XML sloky.

3.1.2.1 XML datový tok

XML datový tok slouží jako nosič pro vzájemnou výměnu XML elementů mezi dvěma entitami v síti. Začátek XML toku je jednoznačně označen otevírací XML značkou tzv. „hlavičkou toku“ (XML značka <stream> s požadovanými atributy a deklaracemi jmenných prostorů). Konec XML toku je jednoznačně označen uzavírací XML značkou </stream>. Mezi těmito dvěma značkami je datový tok aktivní a entity si jeho prostřednictvím mezi sebou mohou vyměňovat neomezený počet XML elementů: elementy sloužící jak k vyjednání vlastností toku (např. ověřování SASL mechanizmem nebo šifrování pomocí TLS), tak i tzv. XML sloky. XML tok je iniciován ze strany zakládací entity k přijímací entitě obvykle ze strany klienta na server a odpovídá připojení klienta k serveru.

```
I: <?xml version='1.0'?>
   <stream:stream
       from='juliet@im.example.com'
       to='im.example.com'
       version='1.0'
       xml:lang='en'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'>
```

Takto otevřený tok umožňuje jednosměrnou komunikaci ze zakládací entity k přijímací entitě. K umožnění výměny XML slok z přijímací entity k zakládací entitě je nutné založit tok v opačném směru.

```
R: <?xml version='1.0'?>
   <stream:stream
       from='im.example.com'
       id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
       to='juliet@im.example.com'
       version='1.0'
       xml:lang='en'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'>
```

3.1.2.2 XML sloka

XML sloka je základní jednotkou pro přenos informace a významem se podobá paketu nebo zprávě použitých v jiných síťových protokolech. Sloka je element první úrovně (přímý potomek uzlu <stream/>), který nese název buď „message“, „presence“ nebo „iq“ a nachází se ve jmenném prostoru s názvem „jabber:client“ nebo „jabber:server“. Ostatní elementy v této úrovni s jiným názvem nebo z jiného jmenného prostoru se nenazývají sloky a týkají se řízení samotného datového toku (chyby datového toku, elementy týkající se SASL přihlašování a protokolu TLS apod.) Též elementy s tímto názvem, ale v jiné než první úrovni, se nenazývají XML sloky. XML sloka obvykle obsahuje jeden nebo více vnořených elementů (s patřičnými atributy, elementy a znakovými daty) pro vyjádření dané informace a může patřit do nějakého jmenného prostoru.

3.1.3 XML sloky

V následujících podkapitolách je uveden stručný popis druhů XML slok. Je zde uvedeno pro jaké účely se používají a jaký nesou význam.

3.1.3.1 Message

XMPP sloka <message/> je základní „push“ metoda k doručení informace z jednoho místa na druhé. Zprávy obvykle nemají potvrzené doručování, a proto se používají k rychlému šíření informace z jednoho místa na jiné. Tyto sloky se používají obzvláště v IM aplikacích, aplikacích pro hromadné chatování, pro zasílání různých upozornění nebo informačních zpráv a podobně. Sloka message se vyskytuje v několika variantách, které jsou od sebe rozlišené atributem *type*. Sloka navíc obsahuje atributy *to* a *from* a může obsahovat atribut *id*, který se používá k rozlišení jednotlivých zpráv a udržování kontextu (atribut *id* se používá hlavně v IQ slokách). Atribut *to* obsahuje Jabber ID

adresáta dané zprávy a atribut *from* obsahuje Jabber ID odesilatele. Atribut *from* není doplněn klientem, ale vkládá ho až server odesilatele, aby se zabránilo možnému podvržení adresy. Zpráva může také obsahovat i další vnořené elementy. Základní specifikace XMPP definuje některé nejákladnější elementy, jako jsou například <body/> a <subject/>, zpráva ale může obsahovat i jiné zde nedefinované elementy.

Normal

Sloky message s atributem type s hodnotou *normal* jsou nejvíce podobné emailovým zprávám. Jsou to samostatné zprávy, na které může nebo nemusí být očekávána odpověď.

Chat

Sloky message s atributem type s hodnotou *chat* se používají ke komunikaci v real-time relaci mezi dvěma entitami, například při rozhovoru dvou přátel v IM aplikaci.

Groupchat

Sloky message s atributem type s hodnotou *groupchat* se používají ke komunikaci ve více-uživatelských IM aplikacích, podobné aplikaci IRC.

Headline

Sloky message s atributem type s hodnotou *headline* se používají k zasílání upozornění nebo oznámení. Na tyto zprávy se neočekává žádná odpověď (klient, který příjme zprávu tohoto typu, by uživateli neměl umožnit odpovědět).

Error

Pokud nastane nějaká chyba, která se vztahuje k předešlé odeslané zprávě, tak entita, která zjistila tuto chybu, zašle odesilateli sloku message s atributem type nastaveným na hodnotu *error*.

3.1.3.2 Presence

Jedním z význačných rysů systémů pro komunikaci v reálném čase je signalizace aktuální síťové dostupnosti. Pro tento účel se používá sloka <presence/>, pomocí které entita oznamuje svoji přítomnost v síti a umožňuje ostatním entitám vědět, zda je entita připojena k síti, a tak dostupná pro real-time komunikaci. Entity v síti nejsou automaticky informovány o přítomnosti dané entity, dokud nemají k této činnosti patřičné oprávnění té dané entity. Pro udělení tohoto oprávnění je nutné, aby entita, která si přeje být informována o síťové dostupnosti určité entity, zaslala dané entitě patřičný požadavek, který musí být tou danou entitou schválen. Jakmile je tento požadavek schválen, entita dostává pravidelná oznámení o síťové dostupnosti dané entity.

Entita může zaslat informaci o své přítomnosti jiné entitě přímo tím, že specifikuje Jabber ID adresáta v atributu *to* elementu <presence/>. Ale obvyklejší je, že entita zašle oznámení o své

přítomnosti XMPP serveru, ke kterému je připojena, a ten odešle toto oznámení o přítomnosti všem entitám, které jsou pověřeny tuto informaci obdržet.

Entita oznamující svoji přítomnost může poskytnout podrobnější informaci o své dostupnosti tím, že do sloky <presence/> vloží element <show/> obsahující řetězec charakterizující danou informaci. Pro ještě detailnější popis své dostupnosti může entita vložit do sloky <presence/> element <status/>, ve kterém může uvést podrobnější popis aktuálního stavu své dostupnosti.

```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <status xml:lang='cz'>Ja dvo&#x0159;&#x00ED;m Juliet</status>
</presence>
```

3.1.3.3 IQ

Sloka IQ neboli *Info/Query* poskytuje mechanizmus pro výměnu informací způsobem dotaz-odpověď. Tento mechanizmus je podobný metodám GET, POST a PUT známých z protokolu HTTP. Oproti sloce <message/> může obsahovat pouze jeden vnořený element, který definuje daný požadavek, který má být zpracován, nebo akci, která má být vykonána příjemcem. Navíc entita, která odešle sloku tohoto druhu, musí vždy obdržet i odpověď (obvykle vytvořenou klientem nebo serverem příjemce). Požadavky a na ně příslušné odpovědi jsou identifikovány pomocí atributu id, jehož hodnota je generována žádající entitou a vložena do odpovědi příslušnou odpovídající entitou. Jednotlivé varianty IQ sloky se od sebe liší hodnotou atributu *type*.

Get

Žádající entita, která požaduje po adresované entitě nějakou informaci, zašle IQ sloku s hodnotou atributu *type* nastavenou na hodnotu *get*. Tato metoda se podobá metodě GET u protokolu HTTP.

Set

Entita, která zaslala IQ sloku s hodnotou atributu *type* nastavenou na hodnotu *set*, poskytuje nějakou informaci nebo tvoří požadavek, který má být zpracován adresovanou entitou. Tato metoda se podobá metodě POST nebo PUT protokolu HTTP.

Result

Entita, která odpovídá na požadavek typu *get* nebo potvrzuje operaci typu *set*, zapouzdří odpověď s požadovanými informacemi do IQ sloky s atributem *type* nastaveným na hodnotu *result*.

Error

Entita, která nedokáže odpovědět nebo zpracovat požadavek typu *get* nebo *set*, oznámí žádající entitě prostřednictvím IQ sloky s hodnotou atributu *type* nastavenou na hodnotu *error*, že nebyla schopna zpracovat daný požadavek (např. byl zaslán vadný požadavek, entita nemá dostatečná práva k provedení dané operace, apod.).

3.1.4 Adresování

XMPP používá ke směrování a doručování zpráv skrze síť globálně unikátní adresy založené na systému DNS, podobně jako email. Adresovatelné jsou všechny XMPP entity, zejména klienti a servery, adresovatelné jsou ale i různé služby, které mohou klienti i servery využívat. Adresy serverů jsou ve tvaru doménového jména (např. im.example.com). Adresy účtů spravované tímto serverem jsou ve tvaru účet@doména (např. user@im.example.com) a nazývají se „prosté JID“. Konkrétní připojené zařízení nebo prostředek, který je autorizovaný prostřednictvím tohoto účtu, má adresu ve tvaru účet@doména/prostředek (např. user@im.example.com/home). Adresa v tomto tvaru se nazývá „úplné JID“. Z historických důvodů se XMPP adresám často říká Jabber ID nebo zkráceně JID. [2]

3.1.5 Oznamování o přítomnosti

XMPP standard definuje schopnost entitě oznamovat svoji síťovou dostupnost nebo přítomnost jiným entitám v síti. Oznamování o dostupnosti probíhá pomocí sloky <presence/>. Znalost síťové dostupnosti není k výměně XMPP zpráv nutná, ale napomáhá k interakci v reálném čase, kdy odesilatel zprávy před zahájením vlastní komunikace ví, zda je příjemce připojený k síti či nikoliv.[2]

Nejzákladnějšími stavy přítomnosti jsou stavy „Připojen“ a „Nepřipojen“. Nicméně standard XMPP umožňuje tyto základní stavy přítomnosti dále rozšířit o další stavy, jako jsou například stavy „Nevyrušovat“ nebo „Pryč“. Tyto stavy lze dále přizpůsobit použitím stavových zpráv, kde je možné uvést například podrobnější informace o aktuálním stavu.

3.1.6 Roster

Termínem roster je v technologii XMPP je označován seznam kontaktů. Skládá se z libovolného počtu položek a každá položka je jednoznačně identifikována pomocí Jabber ID. Seznam kontaktů spojený s určitým účtem je uložen na XMPP serveru, na kterém je daný účet registrován a spravován. K tomuto seznamu kontaktů mají přístup všechny entity, které jsou pomocí daného účtu přihlášeny k síti protokolu XMPP/Jabber.[2]

Seznam kontaktů uložený na XMPP serveru je možné spravovat prostřednictvím IQ slok obsahující element <query/> s definicí jmenného prostoru s názvem *jabber:iq:roster*. Element <query/> může obsahovat jeden nebo více vnořených elementů <item/>, každý popisující jednu položku neboli kontakt v seznamu kontaktů.

Klíč nebo unikátní identifikátor každé položky v seznamu kontaktů je Jabber ID, který je uložen v povinném atributu *jid* elementu <item/>. Stav oznamování o přítomnosti (stav vzájemného oprávnění vědět o své přítomnosti) ve vztahu k dané položce v seznamu kontaktů je uložen v atributu *subscription* elementu <item/>.

Atribut *subscription* může nabývat následujících hodnot: *none,* kdy uživatel a daný kontakt si navzájem nevyměňují informace o své přítomnosti; *to,* kdy uživateli jsou zasílány informace o přítomnosti daného kontaktu, ale uživatel nezasílá danému kontaktu informace o své přítomnosti; *from*, kdy uživatel zasílá danému kontaktu informace o své přítomnosti, ale uživateli nejsou zasílány informace o přítomnosti daného kontaktu; *both,* kdy uživatel a daný kontakt si navzájem posílají informace o své přítomnosti.

Každý element <item/> může obsahovat atribut *name*, jehož hodnota slouží jako přezdívka vztahující se k danému Jabber ID a je určena uživatelem (hodnota není ovlivněna kontaktem). Každý element <item/> může navíc obsahovat jeden nebo více vnořených elementů <group/>, které slouží pro rozdělení položek seznamu kontaktů do různých kategorií.

3.1.7 Rozšiřitelnost

Protokol XMPP je založen výhradně na technologii XML s rozsáhlým využitím jmenných prostorů, které definují význam a rozsah platnosti vnořených elementů. Právě díky vytváření nových jmenných prostorů lze protokol XMPP dále rozšiřovat. Jednotlivá rozšíření jsou identifikována podle názvu elementu a názvu odpovídajícího jmenného prostoru, ve kterém je daný element definován. Rozšíření jsou často publikována XMPP Standards Foundation, ale je možné definovat svoje vlastní privátní rozšíření. Specifikace veřejných rozšíření protokolu XMPP jsou označena zkratkou XEP následované číselným identifikátorem.

V následujících podkapitolách jsou uvedena použitá rozšíření protokolu XMPP při řešení projektu.

3.1.7.1 XEP-0030: Service Discovery

Tato specifikace definuje rozšíření protokolu XMPP pro zjišťování informací o jiných entitách. Je možné zjistit dva druhy informací: identitu a schopnosti dané entity zahrnující výčet protokolů a služeb, které podporuje, a seznam položek, které se vztahují k dané entitě, jako je seznam místností spravovaný službou pro víceuživatelský chat.[2]

K získání takovýchto informací je zapotřebí, aby žádající entita zaslala cílové entitě IQ sloku typu *get* obsahující prázdný element <query/> definovaný ve jmenném prostoru s názvem *http://jabber.org/protocol/disco#info*. Cílová entita musí zaslat jako odpověď buď IQ sloku typu *result* nebo IQ sloku typu *error*. Pokud entita zašle IQ sloku typu result, tak musí obsahovat vnořený

element <query/> s definicí jmenného prostoru s názvem *http://jabber.org/protocol/disco#info*, který dále obsahuje jeden nebo více elementů <identity/> a jeden nebo více elementů <feature/>.

Každý element <identity/> musí navíc obsahovat atributy *type* a *category*, které určují typ a do jaké kategorie daná entita patří. Element může dále obsahovat volitelný atribut *name*, který vyjadřuje název entity.

Element <feature/> musí obsahovat atribut *var*, ve kterém je uložen název jmenného prostoru protokolu nebo služby, kterou daná entita nabízí. Daná entita musí přinejmenším uvést, že podporuje tento samotný protokol tím, že uvede jmenný prostor s názvem *http://jabber.org/protocol/disco#info*.

3.1.7.2 XEP-0199: XMPP Ping

Jak bylo uvedeno výše, XML toky jsou vázány na TCP spojení. Bohužel TCP spojení může být přerušeno, aniž by se o tom aplikační (XMPP) vrstva dozvěděla. Tradičním způsobem, který řešil tento problém, bylo posílat tzv. „whitespace pings" přes XML tok. Tato specifikace doporučuje více-sofistikovanější přístup, který je přívětivější k technologii XML.[2]

Protokol rozšíření XMPP Ping je velmi jednoduchý. Entita, která si přeje zjistit stav spojení s jinou entitou, zašle dané entitě IQ sloku typu *get* obsahující element <ping/>, který je definován ve jmenném prostoru s názvem *urn:xmpp:ping*. Odpovídající entita zašle buď IQ sloku typu *result*, pokud tento protokol podporuje, nebo IQ sloku typu *error*, pokud nikoliv.

3.1.7.3 XEP-0047: In-Band Bytestreams

Tato specifikace definuje rozšíření protokolu XMPP s názvem In-Band Bytestreams (IBB), které umožňuje dvěma entitám vytvořit mezi sebou virtuální datový tok, přes který si mohou vyměňovat fragmenty dat zakódované algoritmem Base64. Jedná se o datový tok pro obecné použití, a proto je jím možné přenášet data různého charakteru. Doposud se používá IBB jako záložní metoda pro přenos souborů v případě, kde není možné použit jinou metodu, jakou je například SOCKS5 Bytestreams, definovanou ve specifikaci rozšíření XEP-0065. Nicméně metoda IBB může být užitečná v aplikacích s relativně nízkými nároky na množství přenášených dat, jako jsou hry nebo šifrovaný text.

K vytvoření datového toku metodou IBB je zapotřebí, aby iniciátor poslal IQ sloku typu *set* obsahující vnořený element <open/>, který je definovaný ve jmenném prostoru s názvem *http://jabber.org/protocol/ibb*. Tento element má 3 atributy: povinný atribut block-size, který určuje maximální velikost přenášeného fragmentu dat v bytech; povinný atribut sid, který tvoří jednoznačný identifikátor této IBB relace; volitelný atribut stanza, který určuje, zda se data budou posílat pomocí IQ slok nebo slok <message/> (implicitně se pro přenos používají IQ sloky). Pokud si adresát přeje pokračovat a vytvořit IBB relaci, zašle iniciátorovi IQ sloku typu *result*. Pokud si adresát naopak nepřeje pokračovat nebo toto rozšíření nepodporuje, zašle iniciátorovi IQ sloku typu *error* s patřičnými informacemi o chybě.

Pokud adresát informuje iniciátora, že si přeje v relaci pokračovat, iniciátor může začít posílat data přes vytvořený datový tok (protože je tento datový tok obousměrný, tak adresát může taktéž začít posílat data). Každý zaslaný fragment dat je zapouzdřený v elementu <data/>, který je definovaný ve jmenném prostoru s názvem *http://jabber.org/protocol/ibb*. Tento element by měl být posílán v IQ sloce, aby bylo zajištěno potvrzování přijetí a řízení zahlcení. Velikost dat nesmí přesáhnout velikost uvedenou v atributu *block-size*, která byla vyjednána při vytváření relace. U elementu <data/> musí být uveden atribut *seq*, který funguje jako čítač odeslaných fragmentů dat určitým směrem. Hodnota atributu *seq* začíná od 0 a je inkrementována po každém odeslaném fragmentu danou entitou. Element <data/> musí dále obsahovat i atribut *sid,* pomocí kterého lze identifikovat, do jaké IBB relace daný fragment patří.

K uzavření datového toku je zapotřebí, aby jedna ze zúčastněných entit poslala té druhé IQ sloku typu *set* obsahující element <close/>. Druhá strana musí uzavření datového toku potvrdit zasláním IQ sloky typu *result.*

3.1.8 Schéma komunikace

1) Určit IP adresu a port k navázání spojení typicky založeném na překladu plně specifikovaném doménového jména
2) Navázat TCP spojení
3) Otevřít XML tok na založeném TCP spojení
4) Pokud možno vyjednat TLS (Transport Layer Security) k šifrování komunikačního kanálu
5) Ověřit klienta pomocí SASL (Simple Authentication and Security Layer) mechanizmu
6) Svázat prostředek a daný XML tok
7) Vyměnit si libovolný počet XML slok s ostatními entitami na síti
8) Uzavřít XML tok
1) Uzavřít TCP spojení

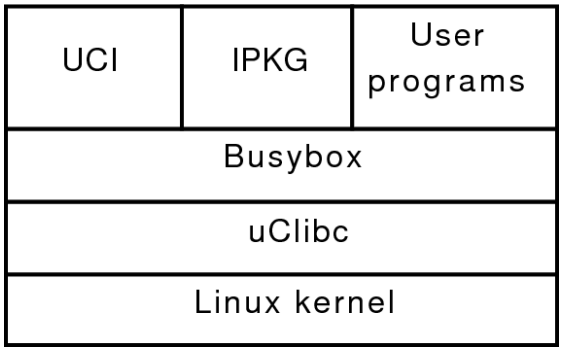
3.2 Iksemel

Iksemel je knihovna funkcí pro zpracování dat ve formátu XML, která byla navržena především pro XMPP aplikace. Obsahuje podprogramy pro práci se SAX, DOM a další funkce specifické pro XMPP protokol. Knihovna je kompletně napsána v jazyce ANSI C a síťový kód je kompatibilní s normou POSIX. Díky svým nízkým paměťovým nárokům je vhodná pro vestavěná zařízení. Obsahuje podporu pro přihlašování pomocí mechanizmu SASL a šifrování pomocí kryptografického protokolu TLS k zabezpečení protokolu XMPP/Jabber. Autorem knihovny je Gurer Ozen. Knihovna byla vydána pod licencí LGPL.[3]

3.3 OpenWrt

OpenWrt je GNU/Linux distribuce určená pro vestavěná zařízení. Na rozdíl od statické formy firmwaru, kde je veškerá funkčnost zakomponována už od výrobce bez možnosti jakékoli úpravy, OpenWrt obsahuje základní funkčnost s podporou dynamického přidávání dalších rozšíření díky vlastnímu souborovému systému a správě balíčků, známých z jiných linuxových distribucí. Pro uživatele to znamená, že si může firmware přizpůsobit podle svých vlastních potřeb, a to přidáním dalších či odebráním jiných, pro něj nepotřebných, balíčků. Pro vývojáře to znamená, že se může zaměřit na určité balíčky, aniž by musel testovat a vydávat kompletní firmware.[4]

V následujících podkapitolách je uveden popis hlavních komponent operačního systému OpenWrt.



Obrázek 1. Architektura operačního systému OpenWrt [5]

3.3.1 Linux kernel

Linux kernel tvoří jádro operačního systému OpenWrt. Každé vestavěné zařízení používá specifické jádro.

3.3.2 uClibc

uClibc je knihovna jazyka C určená pro vývoj aplikací pro vestavěná zařízení s firmwarem založeném na linuxovém jádře. Tato knihovna je daleko méně paměťově náročná než glibc, standardní knihovna jazyka C. Na rozdíl od glibc, která podporuje všechny důležité standardy jazyka C na velkém množství různého hardwaru a platforem, tak uClibc se zaměřuje výhradně na vestavěná zařízení s OS založeném na linuxu. Repertoár funkcí knihovny lze dále měnit podle dalších požadavků na paměť. Knihovna je šířena pod licencí LGPL.[6]

3.3.3 BusyBox

BusyBox je minimalistický soubor UNIXových utilit. Poskytuje náhradu za většinu běžných utilit, které jsou k nalezení v různých GNU balíčcích. Utility mají obvykle méně možností, nicméně byly zanechány všechny důležité funkce, které se chovají jako GNU originály. BusyBox byl vytvořen s

ohledem na paměťové nároky a omezené zdroje, a proto je vhodný pro použití ve vestavěných zařízeních. Soubor utilit je modulární, jednotlivé funkce se dají v průběhu kompilace přidávat či odebírat podle paměťových nároků. Projekt je šířen pod licencí GPLv2.[7]

3.3.4 UCI

UCI je zkratka pro Unified Configuration Interface. Je to knihovna jazyka C, která umožňuje konfigurační kontext pro uživatelskou a systémovou konfiguraci a správu. Knihovna může být použita jak pro tvorbu konfiguračního úložiště pro nové aplikace, které je kompatibilní s OpenWrt, nebo ji lze snadno integrovat do již stávající aplikace. Nové verze UCI obsahují webové rozhraní a SNMP rozšíření, skrze které lze snadno přistupovat ke konfiguraci a správě vestavěného zařízení.[5]

3.3.5 Ipkg/opkg

Opkg je odlehčený systém správy balíčků. Je napsán v jazyce C a ovládáním se podobá dpkg, balíčkovacímu systému v linuxové distribuci Debian. Je určen pro vestavěná zařízení s operačním systémem založeném na Linuxu. Používá se v OpenEmbedded a OpenWrt projektech. Jeho vývoj začal odštěpením od projektu ipkg ve spolupráci s Openmoko projektem. Opkg zůstává zpětně kompatibilní s ipkg.[8]

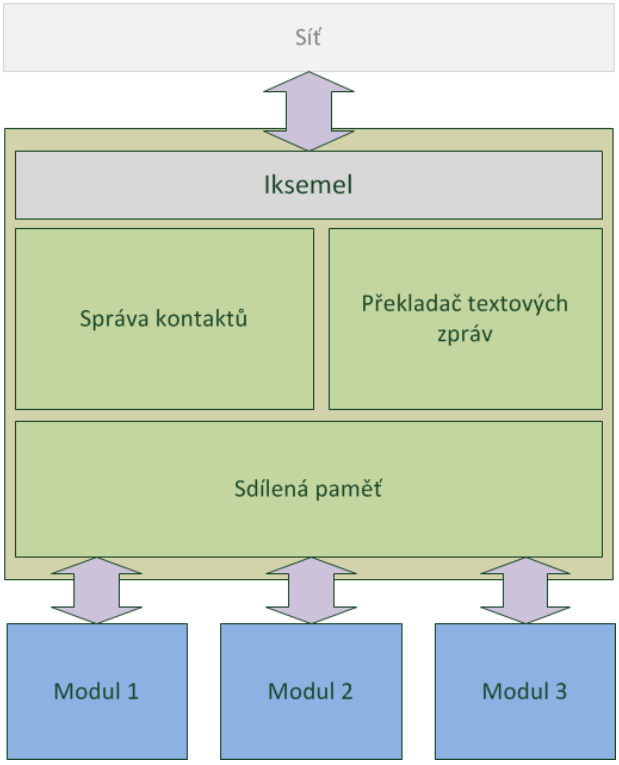
V aktuální verzi OpenWrt byl ipkg nahrazen balíčkovacím systémem opkg.

3.3.6 Uživatelské programy

Uživatelské programy tvoří programovou výbavu operačního systému OpenWrt a definují funkčnost vestavěného zařízení.

4 Návrh

Při návrhu je třeba dbát na to, že aplikace bude nasazena ve vestavěných zařízeních, která budou mít k dispozici omezené výpočetní zdroje, obzvláště kapacitu paměti. Proto je žádoucí, aby výsledná aplikace měla co nejmenší nároky na hardware a neovlivňovala tak negativně celý systém. Další vlastností by měla být modularita a s tím související možnost vypnout nepoužívané součásti k dalšímu snížení nároků. Z požadavků plyne, že aplikace má být schopna interpretovat příkazy a získávat informace o systému, nicméně to neznamená, že aplikace musí mít pouze tuto funkčnost. Díky modularitě a jednotnému způsobu komunikace klient - modul, je možné vytvářet další moduly s různorodou funkčností a pokrýt tak širší spektrum požadavků.



Obrázek 2. Struktura aplikace

4.1 XMPP klient

Jádrem aplikace bude XMPP klient, postavený na knihovně iksemel. Ten bude zajišťovat přijímání a odesílání zpráv a správu kontaktů. Klient musí umožňovat správu předem neurčitého počtu různých příkazů. Dále by měl obsahovat i podprogramy pro přenos souborů. Aplikace bude dále poskytovat i podsystém pro správu zásuvných modulů a každý zásuvný modul aplikace bude disponovat vlastní jednoznačnou identifikací. Komunikace mezi jádrem klienta a moduly bude probíhat pomocí volání specifických funkcí.

4.2 Zásuvné moduly

Zásuvné moduly slouží k rozšíření základní funkčnosti XMPP klienta. V modulech jsou uloženy definice příkazů a implementace funkcí potřebných pro splnění požadované funkčnosti. Díky modulární architektuře je možné přidávat nebo odebírat funkce, kterými XMPP klient disponuje a přizpůsobit jej tak potřebám dané aplikace. Modul bude ve formě dynamicky linkované knihovny, která se nalinkuje při spuštění aplikace.

4.2.1 Modul „Sledování změn v souboru“

Funkce toho modulu budou sloužit k monitorování změn v textovém souboru a ke správě seznamu odběratelů těchto informací. V případě, nastane-li nějaká změna v souboru, se odešle zvoleným adresátům zpráva s informacemi o dané změně. Tento modul je užitečný například pro sledování změn v souborech s různými systémovými či aplikačními záznamy.

4.2.2 Modul „Interpret příkazů“

Interpret příkazů bude poskytovat možnost vzdáleně spouštět jiné programy a bude umožňovat jednoduchou správu interpretem spuštěných procesů. V našem případě tu budou hlavně příkazy shellu operačního systému OpenWrt. Implementace bude založena na funkcích z rodiny funkcí *exec*. Modul je užitečný hlavně v případech, kdy není možné použít jiné prostředky pro vzdálenou správu, jakými jsou např. programy *telnet* nebo *ssh*.

4.2.3 Modul „Plánovač“

Plánovač umožní naplánovat vykonání příkazů na určitou dobu. Plánované příkazy mohou být i příkazy z jiných modulů. Tato funkce je užitečná v případě, že je nutné provést nějakou akci v určitý čas, ale není možné ji v tu danou chvíli provést přímo ručním zadáním patřičného příkazu. Modul bude obsahovat časovač, který bude nastaven na čas nejbližší následující události, a kalendář. Kalendář bude implementován jako seznam, kde jednotlivé události budou seřazeny podle času. Při aktivaci časovače se vyjme z kalendáře daná událost, provede se a časovač se nastaví na následující událost v kalendáři.

5 Implementace

Při vlastní implementaci se vycházelo z návrhu a byly použity technologie a postupy, které jsou uvedeny v předchozích kapitolách. Tato kapitola je rozdělena do dvou částí, na část týkající se popisu implementace klienta protokolu XMPP/Jabber a část týkající se popisu implementace jednotlivých zásuvných modulů.

5.1 XMPP klient

Jak bylo uvedeno již v návrhu tak implementace XMPP klienta, byla napsána v jazyce C a jako standardní knihovna jazyka C byla použita knihovna uClibc. Při implementaci samotného XMPP/Jabber protokolu byly dále použity podprogramy z knihovny iksemel.

Popis implementace XMPP klienta je rozdělen do několika logických celků, a to podle důležitých součástí aplikace.

5.1.1 Schéma běhu aplikace

1) Inicializace
2) Zpracování parametrů příkazové řádky
3) Načtení konfigurace z externího souboru
4) Nalezení a načtení zásuvných modulů
5) Připojení se k síti protokolu XMPP/Jabber
6) Přijímání a odesílání zpráv a zpracovávání příkazů
1) Odpojení se od sítě, uvolnění zásuvných modulů a ukončení aplikace

5.1.1.1 Inicializace

Při spuštění aplikace se nejdříve inicializují globální struktury, kde se posléze ukládají informace potřebné pro běh aplikace. Ukládají se zde údaje nutné k přihlášení do XMPP sítě, kterými jsou Jabber ID a příslušné heslo. Pak se zde dále ukládají cesty k externím souborům: cesta k adresáři obsahující zásuvné moduly, cesta ke konfiguračnímu souboru a cesta k textovému souboru obsahující záznamy o běhu aplikace. Dále se inicializují datové struktury pro uložení dat reprezentující seznam kontaktů a seznam dostupných příkazů.

5.1.1.2 Zpracování parametrů příkazové řádky

Po inicializaci aplikace se provádí vyhodnocení parametrů příkazové řádky. Tyto parametry nejsou povinné a ovlivňují spouštění a běh aplikace.

Volitelným parametrem *c*, následovaném cestou ke konfiguračnímu souboru, lze zvolit, jaká konfigurace má být použita pro aktuální běh programu, implicitně se používá konfigurace uložená v souboru */etc/xmppclient/config.xml*. Dalším volitelným parametrem *l*, následovaný cestou k souboru s aplikačními záznamy, lze zvolit, do jakého souboru se budou ukládat záznamy o běhu aplikace. Implicitně se aplikační záznamy ukládají do souboru *run.log* v adresáři s programem. Volitelným parametrem *tls* je možné aktivovat šifrování komunikačního kanálu kryptografickým protokolem TLS, implicitně se komunikace nešifruje. Pomocí parametru *sasl* lze zvolit možnost zabezpečeného přihlašování metodou SASL, implicitně není přihlašování zabezpečeno a přihlašovací údaje se posílají na server ve formě prostého textu. Zadáním parametru *h* je možné vypsat stručné informace o programu.

5.1.1.3 Načtení konfigurace

Ve fázi načítání konfigurace se aplikace pokusí otevřít zvolený soubor a načíst z něho potřebná data pro běh aplikace. Konfigurace je v aplikaci uložena v příslušných globálních strukturách inicializovaných v předchozí fázi. Program prozatím podporuje načítání konfiguračních souborů ve formátu XML, kde mohly být s výhodou použity podprogramy z knihovny iksemel.

Údaje uvedené v konfiguračním souboru jsou pro běh aplikace nutné, pokud se nepodaří správně načíst jeden nebo více údajů, tak aplikace není schopna pokračovat v běhu (aplikace vytvoří příslušný záznam v souboru aplikačních záznamů a ukončí se).

5.1.1.4 Nalezení a načtení zásuvných modulů

Ve fázi nalezení a načtení zásuvných modulů se aplikace pokusí otevřít adresář se zásuvnými moduly, zadaný cestou uvedenou v konfiguračním souboru, který byl načten v předchozí fázi. Pokud se adresář se zásuvnými moduly nepodaří otevřít (neplatná cesta k adresáři, nedostatečná oprávnění), tak není možné pokračovat v načítání modulů (aplikace vytvoří příslušný záznam v souboru aplikačních záznamů). Aplikace bude pokračovat dál v běhu, nicméně s omezenou funkčností.

Současně při načítání zásuvných modulů se v aplikaci mohou registrovat i nové příkazy, které jsou definovány v příslušných modulech.

5.1.1.5 Připojení se k síti protokolu XMPP/Jabber

Po načtení zásuvných modulů a registraci příkazů se aplikace pokusí připojit k XMPP/Jabber síti prostřednictvím serveru, jehož doménový název se získá z úplného Jabber ID uvedeného v konfiguračním souboru.

Před samotným pokusem o připojení se nejprve inicializuje syntaktický analyzátor protokolu XMPP/Jabber, který je implementovaný v použité knihovně iksemel. Následně se validuje a posléze zpracuje Jabber ID určené pro přihlášení se do XMPP/Jabber sítě. U zadaného Jabber ID se prověří, zda obsahuje přinejmenším název účtu a doménový název XMPP/Jabber serveru (tzv. prosté Jabber

ID). Pokud zadané Jabber ID neobsahuje identifikátor prostředku (typické pro tzv. úplné Jabber ID), tak jako identifikátor prostředku se doplní řetězec *Router*.

Po úspěšném zpracování Jabber ID aplikace vytvoří a nastaví filtr pro filtrovací funkci, která slouží ke směrování přijatých slok do specifických podprogramů pro následné zpracování. Tento filtr dokáže rozlišit: IQ sloky týkající se autentizace, IQ sloky obsahující odpověď na požadavek o zaslání seznamu kontaktů uloženém na serveru, <message/> sloky typu *chat*, prostřednictvím kterých probíhá komunikace a posílají se příkazy. Filtr dále rozlišuje sloky podporovaných rozšíření protokolu XMPP/Jabber, které identifikuje podle názvu jmenného prostoru. Jedná se o rozšíření: Service Discovery, In-Band Bytestreams , XMPP Ping.

Po nastavení filtru se aplikace pokusí navázat TCP spojení s XMPP/Jabber serverem na adrese získané z doménové části Jabber ID a portu uvedeném v konfiguračním souboru. Pokud při této operaci nenastane žádná chyba, tak veškerá data přijatá na tomto spojení se předávají funkci, která zajistí vytvoření a řízení XML datového toku pro vzájemnou výměnu XML elementů mezi aplikací a serverem.

V první fázi vytváření XML datového toku dochází k zasílání přihlašovacích údajů na XMPP/Jabber server. Způsob zasílání těchto údajů je ovlivněn parametry zadanými při spuštění programu. Pokud je zvoleno šifrování datového toku pomocí kryptografického protokolu TLS, tak před samotným odesláním přihlašovacích údajů se datový tok zašifruje. Pokud není zvolen parametr pro zabezpečené přihlašování mechanizmem SASL, je vytvořena a odeslána speciální sloka s přihlašovacími údaji v nezabezpečené formě (ve formě prostého textu). Posléze dochází k vyjednávání vlastností XML datového toku. Zde, pokud je zvolen patřičný parametr, je vyjednáno zabezpečené přihlašování mechanizmem SASL a poté zaslány přihlašovací údaje v zabezpečené formě. Při vyjednávání může server požádat o svázání XML datového toku s identifikátorem prostředku, který serveru slouží k identifikaci jednotlivých datových toků při vícenásobném připojení pomocí stejných přihlašovacích údajů.

Po úspěšném ověření zadaných přihlašovacích údajů je XMPP serveru zaslána hlavička datového toku, a tím je XML datový tok založen. V této fázi je aplikace prostřednictvím tohoto datového toku připravena přijímat XML sloky. V tuto chvíli aplikace XMPP serveru zašle IQ sloku typu *get* s požadavkem o zaslání seznamu kontaktů. Ihned poté klient XMPP serveru zašle také <presence/> sloku s elementem <show/>, který obsahuje řetězec *available*, a tím informuje server o své přítomnosti.

Pokud ověření zadaných přihlašovacích údajů na XMPP serveru neproběhlo úspěšně (chybně zadané Jabber ID nebo heslo), tak klient nemůže pokračovat v běhu, uzavře navázané TCP spojení, vytvoří nový záznam o chybě a přidá jej do aplikačního záznamu aplikace.

Jakmile aplikace odešle požadavek o zaslání seznamu kontaktů, tak očekává kladnou odpověď (IQ sloku typu *result*) obsahující požadovaná data. Po jejich přijetí aplikace prochází přijatý seznam kontaktů a ukládá si do připravených datových struktur ty kontakty, u kterých bylo uděleno vzájemné

pověření přijímat oznámení o své přítomnosti (element <item/> s atributem *subscription* obsahující hodnotu *both*). Protože je pro udělení tohoto oprávnění potřeba souhlas klienta i daného kontaktu, tak ověřením tohoto oprávnění klient identifikuje kontakty, se kterými má pověření komunikovat.

5.1.1.6 Přijímání a odesílání zpráv a zpracovávání příkazů

Po úspěšném připojení a přihlášení se k síti protokolu XMPP/Jabber aplikace odesílá a přijímá zprávy od důvěryhodných subjektů. U zpráv aplikace rozlišuje, zda se jedná o příkaz nebo o pouhou zprávu.

Samotné přijímaní a odesílání zpráv je zajištěno čtením a zápisem do vytvořeného datového toku. K implementaci této činnosti jsou použity podprogramy z knihovny iksemel.

5.1.1.7 Odpojení se od sítě, uvolnění zásuvných modulů a ukončení aplikace

Do této fáze přejde aplikace, pokud obdrží pokyn k ukončení nebo je k ukončení donucena vzniklou chybou.

Před samotným ukončením klient nejprve zašle XMPP serveru sloku <presence/> s vnořeným elementem <show/> obsahující řetězec *unavailable*. Tím klient XMPP serveru dává najevo, že již nebude nadále dostupný pro komunikaci. Pak klient uzavře navázané TCP spojení, odstraní filtr pro třídění přijaté komunikace a uvolní se paměť alokovaná syntaktickým analyzátorem protokolu XMPP/Jabber, které byly vytvořeny při navazování spojení k XMPP serveru. Poté klient postupně zavolá funkce sloužící k uvolnění alokovaných zdrojů jednotlivými zásuvnými moduly a uzavře je. Klient následně uvolní globální datové struktury vytvořené při inicializaci aplikace. Nakonec klient vytvoří a přidá patřičný záznam do souboru s aplikačními záznamy aplikace a ukončí se.

5.1.2 Příkazy

Aplikace je ovládána textovými příkazy, které jsou přijímány od pověřených entit připojených k síti protokolu XMPP/Jabber prostřednictvím textových zpráv. Příkazy jsou registrovány během spouštění aplikace nebo načítání jednotlivých modulů.

Registrací příkazu je nazvána činnost, při které se do seznamu dostupných příkazů přidá nová položka, ve které je uložen název daného příkazu a ukazatel na podprogram, kde je daný příkaz definován.

V aplikaci jsou definovány některé základní příkazy, které jsou určeny pro ovládání samotné aplikace. Tyto příkazy slouží k ukončení aplikace, získání seznamu dostupných příkazů, odeslání zprávy nějaké jiné entitě v síti a nastavení režimu dostupnosti v síti protokolu XMPP/Jabber.

Dále je možné vytvářet aliasy a tím zjednodušit zadávání složitějších příkazů podobně jako v příkazovém interpretu *bash*. Aliasy se vytvářejí příkazem *alias* následovaný názvem vytvářeného aliasu a vlastním příkazem. Zrušení existujícího aliasu se provádí příkazem *unalias*. Aliasy jsou

ukládány v konfiguračním souboru, aby byly k dispozici i po ukončení a opětovném spuštění aplikace.

5.1.3 Systém zásuvných modulů

Systém zásuvných modulů je implementován prostřednictvím dynamicky linkovaných knihoven, které jsou napsány v jazyce C. Zásuvné moduly jsou uloženy ve zvláštním adresáři a cesta k tomuto adresáři je uložena v konfiguračním souboru aplikace.

5.2 Modul „interpret příkazů“

Zásuvný modul „interpret příkazů“ obsahuje implementace příkazů pro vzdálené spouštění jiných programů. Modul si ukládá informace i jím spuštěných procesech, aby umožnil jejich další řízení. Bylo třeba zajistit možnost běhu více paralelních procesů spuštěných více uživateli. Implementace je založena na POSIX vláknech a funkcích z rodiny *exec* a funkci *fork*.

5.2.1 Řízení procesů

Modul si ukládá v připravených datových strukturách informace o jím spuštěných procesech. Pomocí těchto informací je možné sledovat jejich stav a dodatečně je řídit.

5.2.2 Schéma činnosti modulu

1) Inicializace
2) Spouštění, řízení a ukončování procesů
1) Ukončení činnosti a uvolnění zásuvného modulu

5.2.2.1 Inicializace modulu

Při načtení modulu aplikací dochází k inicializaci datových struktur potřebných pro činnost modulu.

Vytváří se a inicializuje seznam procesů, kde se ukládají informace o spuštěných procesech. Dále dochází k registraci příkazů, které modul implementuje.

5.2.2.2 Spouštění, řízení a ukončování procesů

Jakmile je zadán příkaz ke spuštění procesu, tak je vytvořen nový záznam s informacemi o procesu, kde se nejprve uloží údaje jako je Jabber ID zadávající entity, název spuštěného programu a jeho parametry. Poté se vytvoří nezávislé vlákno, kterému se daný záznam předá.

V nově vytvořeném vláknu se inicializují roury, které budou sloužit k přesměrování standardního a chybového výstup do předem připravené vyrovnávací paměti. Poté se zavolají funkce *fork* a *execve* , které společně vytvoří nový proces. Do záznamu o procesu se vloží identifikátor (PID)

nově vytvořeného procesu a celý záznam se vloží do seznamu procesů. Pak v režii vlákna probíhá čtení z vyrovnávací paměti pro výstup procesu a přečtená data se posílají jako zprávy zadavateli příkazu. Zprávy se posílají vždy, když se vyrovnávací paměť naplní nebo proces uzavře svůj výstup. Jakmile se daný proces ukončí, odebere se patřičný záznam procesu ze seznamu procesů a vlákno spojené s daným procesem se ukončí.

Běžící procesy lze řídit nebo ukončovat pomocí příkazu pro zasílání signálů určeným procesům.

5.2.2.3 Ukončení činnosti a uvolnění zásuvného modulu

Jakmile je aplikací zavolána funkce pro uvolnění alokovaných zdrojů zásuvným modulem, jsou ukončeny všechny běžící procesy a s nimi spojená vlákna a uvolněna paměť alokována seznamem procesů.

5.3 Systémové nároky výsledné aplikace

Z požadavků plyne, že aplikace má být spustitelná na směrovačích vybavených operačním systémem OpenWrt, nicméně návrh a následná implementace umožňují spustit aplikaci i na jiných zařízeních s operačním systémem založeném na Linuxu. Aplikace pro svůj běh vyžaduje nainstalovanou knihovnu iksemel a pro volitelnou podporu šifrování komunikace vyžaduje nainstalovanou i knihovnu gnutls.

Pro budoucí rozšíření by klient napsán tak, aby byl co nejvíce platformě nezávislý, ale zároveň bylo možné díky své modulární architektuře vytvářet zásuvné moduly s využitím funkcí specifických pro konkrétní platformu.

Níže jsou uvedeny orientační tabulky obsahující statistiky využití virtuální paměti aplikací. Tyto statistiky byly získány pro aplikaci běžící na směrovači Netgear WNDR3700v2 s operačním systémem OpenWrt a pro aplikaci běžící na osobním počítači s operačním systémem Arch Linux (x86). Jsou zde uvedeny dále statistiky pro klienta bez zásuvných modulů a pro klienta s modulem „Interpret příkazů“. Údaje byly získány ze souboru *status* v podadresáři daného procesu v adresáři */proc*, který slouží jako rozhraní operačního systému pro získávání informací o běžících procesech.

VmExe - využití virtuální paměti spustitelným souborem a staticky linkovanými knihovnami

VmLib - využití virtuální paměti dynamicky linkovanými knihovnami

VmStk - využití virtuální paměti zásobníkem procesu

VmData - využití virtuální paměti alokované na haldě procesu

OpenWrt	VmExe (kB)	VmLib (kB)	VmStk (kB)	VmData (kB)	VmSize (kB)
Klient	28	1420	88	532	2136
Klient s modulem	28	1526	88	540	2280

Tabulka 1Využití virtuální paměti – OpenWrt

Arch Linux	VmExe (kB)	VmLib (kB)	VmStk (kB)	VmData (kB)	VmSize (kB)
Klient	28	1716	136	172	2116
Klient s modulem	28	1820	136	180	2240

Tabulka 2Využití virtuální paměti - Arch Linux

Ze získaných hodnot lze usoudit, že největší část virtuální paměti obsazené aplikací okupují dynamicky linkované knihovny (iksemel a gnutls) a v porovnání s ní je část obsazená spustitelným souborem se staticky linkovanými knihovnami relativně malá. Rovněž je zde vidět nárůst velikosti virtuální paměti obsazené dynamicky linkovanými knihovnami při použití zásuvného modulu „Interpret příkazů“. Když porovnáme hodnoty z pohledu platforem, zjistíme, že velikost využité virtuální paměti aplikací je podobná.

Mezi obsluhováním jednotlivých příkazů je proces aplikace v režimu *idle* a tudíž výrazně nezatěžuje procesor. Při spuštění procesu pomocí příkazového interpretu zátěž odpovídá spouštěnému procesu.

6 Závěr

Výsledkem této bakalářské práce je modulární klient protokolu XMPP/Jabber pro interpretaci příkazů určený pro operační systémy založené na Linuxu, včetně operačního systému OpenWrt. Díky modulární architektuře je možné funkčnost klienta přizpůsobit různým požadavkům aplikace. Spolu s klientem byl implementován také modul pro interpretaci příkazů shellu operačního systému, pomocí něhož je možné spravovat dané zařízení a získávat informace nejen o síťovém provozu. Modul je užitečný hlavně v případech, kdy není možné použít jiné prostředky pro vzdálenou správu.

Součástí návrhu aplikace jsou i zásuvné moduly „Plánovač“ a „Sledování změn v souboru“, které nebyly prozatím implementovány. Jejich implementace je součástí plánů do budoucna. Stejně tak je součástí těchto plánů i implementace přenosu souborů, která se nezdařila, protože původní implementace se ukázala být nekompatibilní s většinou ostatních klientů protokolu XMPP/Jabber a je nutné implementovat další podpůrné prostředky.

Tématu této bakalářské práce se chci věnovat i nadále a stávající aplikaci rozšířit o další prvky, které by zvyšovali její užitnost.

Literatura

[1] SAINT-ANDRE, Peter; SMITH, Kevin; TRONÇON, Remko. *XMPP: The Definitive Guide : Building Real-Time Applications with Jabber Technologies*. First Edition. Sebastopol (CA) : O'Reilly Media, c2009. 285 s. ISBN 978-0-596-52126-4.

[2] The XMPP Standards Foundation. *XMPP Standards Foundation* [online]. 2011 [cit. 2011-01-23]. Dostupné z WWW: <http://xmpp.org/>.

[3] Google. *Iksemel : Fast and portable XML parser and Jabber protocol library* [online]. c2010 [cit. 2011-01-10]. Project Home. Dostupné z WWW: <http://code.google.com/p/iksemel/>.

[4] *OpenWrt : Wireless Freedom* [online]. 2010 [cit. 2011-01-10]. What is OpenWrt?. Dostupné z WWW: <http://openwrt.org/>.

[5] FAINELLI, Florian. The OpenWrt embedded development framework [online]. [s.l.] : [s.n.], 22.1.2008 [cit. 2011-02-01]. Dostupné z WWW: <http://downloads.openwrt.org/people/florian/fosdem/fosdem.pdf>.

[6] ANDERSEN, Erik. *UClibc* [online]. c2010 [cit. 2011-01-10]. A C library for embedded Linux. Dostupné z WWW: <http://www.uclibc.org/about.html>.

[7] ANDERSEN, Erik. *BusyBox* [online]. c2008 [cit. 2011-01-10]. BusyBox: The Swiss Army Knife of Embedded Linux. Dostupné z WWW: <http://busybox.net/about.html>.

[8] Google. *Opkg : Open PacKaGe Management* [online]. c2010 [cit. 2011-02-01]. Project Home. Dostupné z WWW: <http://code.google.com/p/opkg/>.

[9] Geeknet. *SourceForge* [online]. c2011 [cit. 2011-01-22]. Iperf. Dostupné z WWW: <http://sourceforge.net/projects/iperf/>.

Seznam použitých zkratek

DNS (Domain Name System); hierarchický systém doménových jmen

DOM (Document Object Model); objektový model dokumentu

HTTP (Hypertext Transfer Protocol); protokol pro výměnu dokumentů ve formátu HTML

IBB (In-Band Bytestreams); rozšíření protokolu XMPP umožňující přenos dat

IQ (Info/Query); druh XML sloky

MB (MegaByte); jednotka objem dat, 1 048 576 Bajtů

SASL (Simple Authentication and Security Layer); metoda pro ověřování příhlašovacích údajů

SAX (Simple API for XML); funkce umožňující sériový přístup k XML

TCP (Transmission Control Protocol); internetový protokol

TLS (Transport Layer Security); kryptografický protokol

XML (Extensible Markup Language); rozšiřitelný značkovací jazyk

XMPP (Extensible Messaging and Presence Protocol); protokol pro výměnu zpráv a informací o přítomnosti