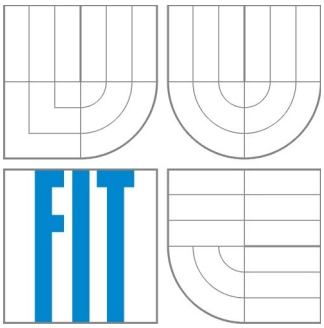
# Vysoké učení technické v Brně

## Fakulta Informačních Technologií



Dokumentace k projektu pro předmět ISA
Ircbot
20. listopadu 2014

Autor: Josef Karásek, xkaras27@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké učení technické v Brně

# Obsah

# 1 Úvod

Tento manuál popisuje jednoduchý IRC bot, tedy program zpracovávající data z IRC serveru. Tato data jsou programem vyhodnocována a pokud nastane určitá událost, program zaznamená zprávu pomocí logovací služby SYSLOG. Popisovaný program byl napsán jako projekt předmětu ISA na FIT VUT v Brně.

## 1.1 IRC

Protokol pro textovou komunikaci mezi procesy. Jelikož se jedná o textová data, je tento protokol používán pro tvorbu programů používaných lidmi ke komunikaci. Každá zpráva obsahuje text a metadata o zdroji zprávy, ukončením jedné zprávy je pak znak konce řádku. Proto se IRC klient někdy nazývá řádkový. IRC služba funguje z pravidla na portu 6667 TCP protokolu.

## 1.2 Syslog

Protokol popisující funkce které logovací server musí implementovat a formát logovacích zpráv na unix-like systémech. Služba SYSLOG pracuje na portu 514, nejčastěji UDP, ale mnoho existujících implementací podporuje i TCP komunikaci.

## 1.3 Sokety

Síťová komunikace je aplikacím nejčastěji zprostředkována pomocí socketů. Sockets API je knihovna zaobalující služby TCP/IP, což vytváří abstrakci nad nízko-úrovňovými operacemi. Knihovna je nejčastěji součástí operačního systému a její funkce jsou využívány skrze systémová volání.

# 2 Návrh

Program je navrhnut s ohledem na přímé využívání nízko-úrovňových prostředků poskytovaných operačním systémem, ale pro zjednodušení práce a snížení chybovosti je přidána další vrstva abstrakce. A to použitím objektového návrhového vzoru Wrapper Facade[3]. Síťové operace jsou zapouzdřeny v třídách, což je Wrapper část. Facade část vzoru je vytvoření veřejně přístupných metod, které definují jednoznačné a transparentní rozhraní pro práci se sockety. Výsledkem je zjednodušené použití jednotlivých systémových volání a snížení rizika zavedení chyb do programu. Například při vytvoření objektu zapouzdřujícího TCP komunikaci je, jak je v objektově orientovaných jazycích zvykem, volán konstruktor – ten zajistí alokaci potřebných prostředků a také inicializaci potřebných datových struktur. Programátor využívající tuto abstrakci nemusí například pamatovat na volání memset() pro inicializaci každé struktury nebo volání close() po ukončení komunikace.

## 2.1 Přenositelnost

Implementace socketů se na různých operačních systémech liší. Například na Unix-like systémech je

socket reprezentován jako file descriptor, tedy signed int. Zatímco ve WinSock systémů Windows je reprezentován jako unsigned int, makrem maskovaný na SOCKET[4]. To vnáší jistou nekompatibilitu při kontrole vytvoření socketu, kdy se v prostředí unix porovnává file descriptor na hodnotu menší než 0, tedy indikátor chyby. Pro limitovanou přenositelnost je ve WinSock chybový stav definován jako bitová negace nuly. Tohle je nicméně pouze demonstrace limitů přenositelnosti aplikací mezi platformami. V praxi se často při psaní multiplatformní aplikací využívajících sockety spoléhá na preprocesor a zdrojový soubor tak obsahuje kód pro více platforem, pro které se pak musí specificky kompilovat.

### 2.2 Model

Tyto informace byly známy již při návrhu aplikace a výsledný model s nimi koresponduje. Sada základních operací, které musí každá implementace vzoru Wrapper Facade obsahovat, je deklarována v čistě virtuální třídě a tu jednotlivé, platformově závislé implementace definují. V případě této aplikace je třeba plně duplexní TCP spojení pro komunikaci s IRC serverem a half-duplexní UDP spojení pro komunikaci se Syslog serverem. Třída SocketNetworkService deklaruje metody, jež třídy, které ji rozšiřují, musí implementovat. Tyto třídy jsou UnixTCPService pro TCP komunikaci a UnixUDPService pro UDP komunikaci. Diagram tříd je dostupný v příloze A. Dále je součástí programu objekt výjimky `NetworkException`, který je určen pro propagaci informací o chybách, které během komunikace mohou nastat.

Soubor utils.cpp obsahuje dvě podpůrné funkce pro práci se `std::string`, `split` pro rozdělení řetězce na podřetězce a `strip` pro odstranění prefixu a postfixu.

## 3 Implementace

Funkce `main()` se nachází v souboru *main.cpp*. V tomto souboru proběhne pouze kontrola argumentů a při korektním spuštění je vytvořen objekt *Ircbot*. Poté je objektu *Ircbot* předána kontrola programu voláním metody `run()`. Celé tělo metody `run()` je vnořeno do *try/catch* bloku. Chyby, které mohou nastat při vytváření spojení nebo během komunikace, jsou detekovány návratovou hodnotou systémového volání nebo informací z IRC serveru. To činí proces distribuce této návratové hodnoty nebo v ideálním případě informace o nastalé chybě velmi složitým. Proto je součástí aplikace výjimka *NetworkException*, jejíž instance je vytvořena při detekci chyby a propagována přes zásobník volání metod až k metodě **run()**, kde je odchytnuta. Poté jsou dealokovány zdroje a program ukončen s chybovým hlášení vypsaným na *stderr* a chybovým kódem.

Algoritmus prohledávání textu přijatého od IRC serveru je obsažen v metodě `dispatchMessage()` a je založen na prvcích knihovny Standart Template Library. Použití překladače GCC může připravit nečekané překvapení při kontrole korektní dealokace paměti programem *valgrind*, kdy je použita jistá optimalizace[5] a i když paměť alokovaná voláními ze STL je dealokována, ukazatele nejsou uvolněny, nýbrž jsou uchovány v cache paměti pro budoucí použití.

## 3.1 Komunikace s IRC

Komunikace je rozdělena na dvě části: přihlášení (sekvenční část) a naslouchání (iterativní část). Během přihlášení jsou serveru poskytnuta potřebná data pro navázání komunikace a v iterativní části jsou v cyklu data čtena. V obou částech jsou přijatá data vyhodnocována a při nastání určité události je odeslána logovací zpráva. Tato událost je dána způsobem spuštění programu. Bez poskytnutí posledního argumentu příkazové řádky jsou logovány všechny zprávy typu *NOTICE a PRIVMSG*. Pokud byl poslední argument poskytnut, jsou logovány pouze zprávy obsahující alespoň jeden element ze zadaného argumentu.

Formát zpráv IRC komunikace umožňuje nezasílat identifikaci. Pokud taková zpráva je určena pro odeslání na logovací server, je jako identifikátor odeslán parametr *hostname* IRC serveru zadaný uživatelem při spouštění programu.

Čtení dat z IRC serveru s sebou nese nežádoucí složitost – předem se nedá určit, kolik dat server pošle. To při čtení ze socketu působí problém, kdy server odešle data, ta jsou během přenosu fragmentována a při jednom čtení ze socketu není zpráva dočtena až do konce. Zbytek zprávy je přečten při další iteraci. Jako důsledek tohoto chování může nastat situace, kdy část zprávy obsahující označení *NOTICE* nebo *PRIVMSG* je zpracována, ale její zbytek ne. Jak je již zmíněno v úvodu, IRC je řádkově orientovaný protokol a i když neznáme délku zprávy, je možné se orientovat podle znaku konce řádku.

## 3.2 PING PONG

Jednou ze zpráv přijatých od IRC serveru je zpráva typu PING. Jde o keep-alive proces, kdy je nutné před vypršením určitého času odpovědět zprávou PONG na adresu ze zprávy PING.

## 3.3 Komunikace se Syslog

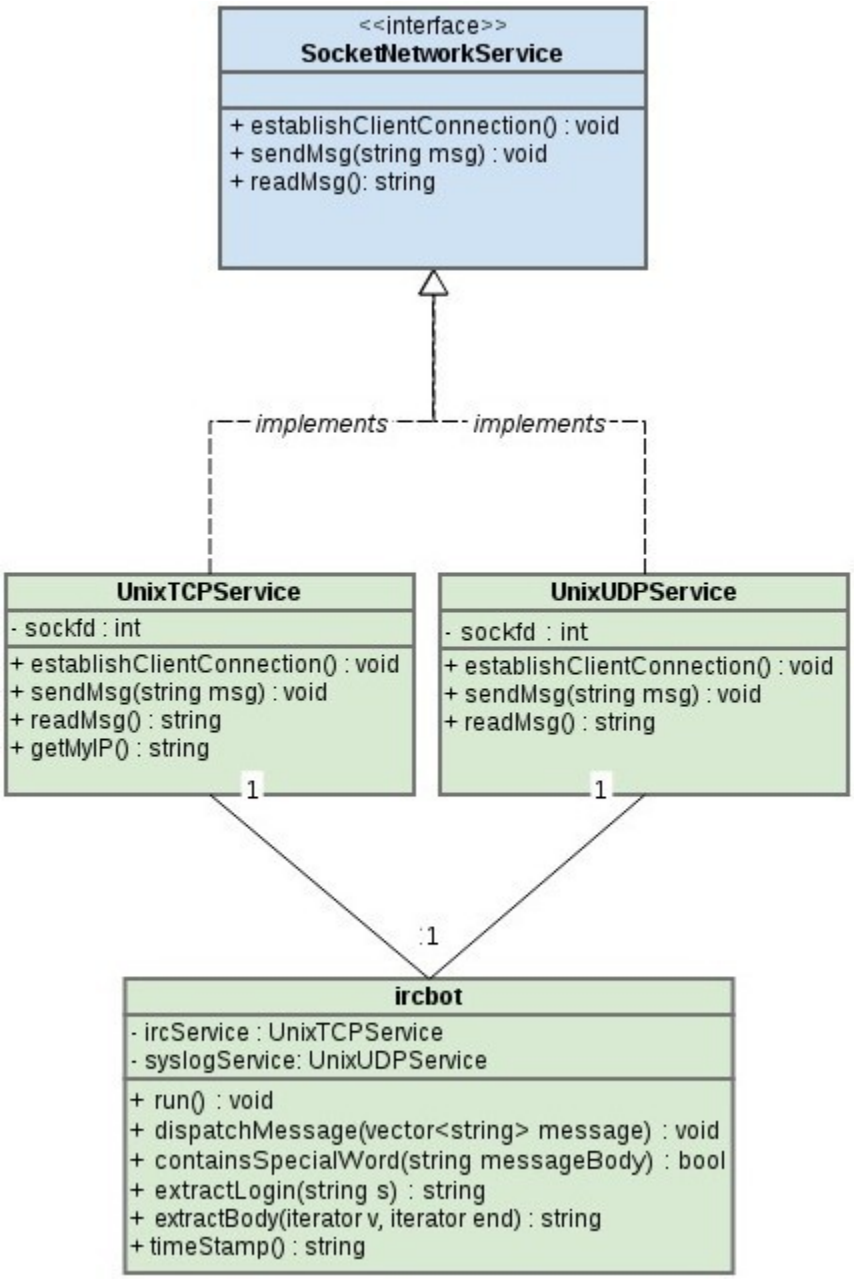
Objekt pro UDP komunikaci je vytvořen před objektem pro TCP komunikaci, a to proto, aby bylo možné logovat informace o případné chybě během sestavování spojení s IRC serverem. Komunikace se Syslog serverem je pouze jednosměrná – odesílání logovacích zpráv. Žádná data ze Syslog serveru přijímána nejsou.

# 4 Závěr

Přestože rozhraní soketů je více než třicet let stará historie, je stále základní implementací TCP/IP stacku a najdeme jej na naprosté většině operačních systémů i když s jistými *nekompatibilitami*. Řešení tohoto projektu si bere za cíl demonstrovat, jak tyto knihovny zakomponovat do návrhu složitějších programů. V dnešní době je síťová komunikace esenciální schopností programu a jsou na ni kladeny nemalé nároky. Ať už se jedná o uživatelské pohodlí, a nebo o doručování zpráv v kritických systémech. Jako zdroj informací pro další studium jsou zajímavé frameworky Ace, Boost a ZeroMQ.

Program byl testován na 64 bitovém systému Fedora 20 a 32 bitovém referenčním stroji Ubuntu 14.04.

# 5 Příloha




*Obr. 1: Diagram tříd*

# 6 Literatura

http://msdn.microsoft.com/en-us/library/windows/desktop/ms740516(v=vs.85).aspx

http://stackoverflow.com/questions/10817252/why-is-invalid-socket-defined-as-0-in-winsock2-h-c

http://valgrind.org/docs/manual/faq.html#faq.reports