VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

ISA – Síťové aplikace a správa sítí

2020/2021

Dokumentace k projektu

Discord bot

Vojtěch Mimochodek (xmimoc01)

18. 11 .2020

Obsah

Úvod

Cílem projektu bylo vytvořit aplikaci (Discord bota), jejíž součástí byl vývoj HTTP klienta, který se připojí k Discord serveru. Následně nalezne uživatelský server s odpovídajícím textovým kanálem. Úkolem bota je naslouchat na daném kanálu a případné zprávy od jiných uživatelů přečíst a zopakovat. To vše je řešeno pomocí příslušných HTTP dotazů zaslaných na Discord API a zpracování příchozích HTTP odpovědí.

Návrh a implementace

Aplikace je rozdělená do různých zdrojových souborů. Každý zdrojový soubor (**.cpp*) má k sobě příslušný hlavičkový soubor (**.h*). Hlavičkové soubory obsahují definice konstant, funkcí, natažení potřebných knihoven a řádné okomentování. Zdrojové soubory pak již žádné další komentáře navíc mimo hlavičku neobsahují.

Aplikace se skládá z těchto souborů:

- main.cpp
- connection.cpp a connection.h
- parse.cpp a parse.h
- input.cpp a input.h
- errors.cpp a errors.h
- requests.cpp a requests.h

MAIN.CPP

Tento soubor obsahuje hlavní logiku programu. Propojuje soubory jako jednotlivé moduly.

Na začátku zavolá funkci *parseArguments()*, která zpracuje vstupní argumenty. V případě správně zadaných argumentů za předpokladu, že nebyl zadán vstupní argument *–h|--help*, tato funkce vrací identifikační token pro bota. Ten byl zadán jako vstupní argument *–t*.

Dále je volána funkce *initConnection()*, která inicializuje SSL spojení pomocí knihovny *OpenSSL* a *Socket*. Po úspěšné inicializaci a navázání spojení je vrácena struktura *connection*, která obsahuje ukazatele na SSL strukturu, CTX strukturu a Socket.

Po inicializacích nastává první zasílání HTTP dotazů na Discord. Prvním dotaz se týká získání clientId, což je ID bota, se kterým pracujeme. Dotaz je zaslán pomocí funkce *sendRequest()*. Z odpovědi v podobně HTTP hlavičky a HTTP těla ve formátu JSON skrze funkci *processResponseJson()* získáme požadované client ID.

Druhým dotazem a použitím stejných funkcí je získáno guildId. Je to v podstatě ID uživatelského serveru na Discordu, jehož je bot členem. Pokud bod není členem žádného serveru, je program ukončen s hláškou a chybovým kódem ***-5***.

Třetí dotaz nám získá ID kanálu, na kterém bot bude dané zprávy opakovat. Je důležité, aby měl kanál pojmenování „isa-bot“. V případě, že server nemá žádný kanál s tímto názvem, je program ukončen s hláškou a chybovým kódem ***-5***.

Pokud jsou všechny předchozí dotazy úspěšné, následuje cyklus *while(true) {...}*. V tomto cyklu se odesílá dotaz, pomocí kterého program zjistí, jestli se v daném kanálu „isa-bot“ nachází nějaké zprávy. V případě, že je kanál prázdný, je v těle odpovědi vráceno „[ ]“. To znamená, že v kanále dosud nebyly žádné předchozí zprávy. V takovém případě, se nastaví proměnná *bool emptyChannel* na hodnotu ***true***. Program se poté uspí na **3** vteřiny. Po uplynutí tohoto času, se opakuje iterace cyklu. Posílá se další dotaz na zjištění zpráv a kontroluje se, zdali jsou k dispozici nové zprávy. Pokud kanál nebyl prázdný, zjistí se ID poslední zprávy v kanále, vystoupí se z tohoto cyklu a vstupuje se do dalšího. Naopak pokud kanál byl prázdný a přišla první zpráva, bot zprávu zopakuje a z cyklu se vystoupí.

Dalším, již zmíněným cyklem, je znovu *while(true) {...}*. Jedná se o hlavní cyklus, který zajišťuje opakování zpráv. Na začátku každé iterace, je odesílán dotaz s *lastMessageID* poslední zprávy. Toto ID je získáno z předchozího cyklu a v dotazu je používáno v příznaku ***?after=*** . To způsobí, že Discord vrátí pouze zprávy, které následovaly po zprávě se zadaným ID. Odpověď, která přišla, je předána funkci *getNewMessages()*. Ta nám vrátí do proměnné *vector<string> newMessages* vector naplněný jednotlivými zprávami. Následuje kontrola, zdali tento vector obsahuje nějaké zprávy. Pokud nám v odpovědi žádné nové zprávy nepřišly, vector s novými zprávami je prázdný. Pokud je prázdný, program se uspí na dobu, která nám přišla v hlavičce odpovědi. Konkrétně v řádku ***x-ratelimit-reset-after***. Zde je hodnota ve vteřinách, za kterou dojde k obnovení limitu HTTP požadavků, které je možno zaslat. Jestliže ale vector není prázdný, vstupuje se do cyklu *for (...) {..}*. V tomto cyklu se postupně projde celý vector od největšího indexu po nejmenší. To proto, aby se zprávy zopakovaly v pořadí, ve kterém byly zaslány uživateli na kanál. Na začátku tohoto for cyklu se z aktuálně zpracovávané zprávy pomocí funkce *getMessageAuthor()* získá ID autora zprávy a jeho jméno. Následuje kontrola, jestli ID autora není shodné s client ID získaným z prvního dotazu. To proto, že bot nesmí opakovat zprávy, které sám zaslal. Také proběhne kontrola, jestli ve jméně autora zprávy není řetězec „bot“. Pokud ano, zprávy od tohoto uživatele nejsou opakovány. Pokud jsou kontroly splněny je ze zprávy získán její obsah skrze funkci *getMessageContent()*. Zpráva je následně poslána zpět do kanálu pomocí HTTP dotazu metodou ***POST***. Když je nastavena globální proměnná *isVerboseActivated* tak je opakovaná zpráva vypsána také na ***STDOUT***. Ze zopakované zprávy je aktualizováno *lastMessageID*. Na konci každé iterace for cyklu, je z aktuálně zpracovávané a zopakované zprávy získán limit počtu dotazů, které je možno ještě zaslat a časový limit, za který dojde k obnově počtu dotazů. K tomu je využívána metoda *getRatelimit()*. Pokud je limit počtu dotazů roven 0, je program uspán na dobu časového limitu, aby došlo k obnově. Počet dotazů, které je možno ještě zaslat je získán z HTTP hlavičky z řádku ***x-ratelimit-remaining***. Poté se pokračuje v dalších iteracích for cyklu, pokud nějaké jsou. Po ukončení tohoto cyklu a tudíž zopakování všech aktuálně zpracovávaných zpráv, je program uspán na dobu

získanou z poslední zopakované zprávy. Poté následuje další iterace while cyklu, získání dalších nových zpráv a jejich zopakování. Tento proces probíhá do té doby, dokud program neukončí uživatel pomocí CTRL+C.

CONNECTION.CPP

V tomto souboru jsou naimplementovány tři funkce.

První funkcí je *initConnection()*. Na začátku provede inicializaci SSL a vytvoření Socketu. Pomocí metody *gethostbyname()* je získána IP adresa ***Discord.com*** na kterou s portem ***443*** proběhne připojení. Funkce vrací zmiňovanou strukturu *connection*.

Druhou funkcí je *clearConnection()*, která uvolní paměť alokovanou pro SSL a CTX struktury a zavře socket.

Poslední funkcí je *sendRequest()*. Jako parametry přijímá ukazatel na SSL strukturu, token bota potřebný ve všech dotazech a v případě konkrétních dotazů také volitelné parametry jako ID guildy/serveru, ID kanálu, ID poslední zprávy nebo také obsah zprávy. Na začátku této funkce jsou volány metody pro sestavení HTTP dotazů. Složené dotazy jsou pomocí metody *SSL_write()* odesílány jako dotazy na Discord. Pomocí metody *SSL_read()* ve *while(true) {...}* cyklu je načítána HTTP odpověď. Odpověď je načítána do *buffer*. Velikost tohoto bufferu je 4096 bajtů. Na konec každého načtení je na poslední pozici vloženo ***'\0'***. Poté je obsah bufferu vložen do *response*. V průběhu každé iterace proběhne kontrola, jestli je odpověď ***chunked***. Pokud ano, z cyklu se vystupuje a odpověď je načítána pouze jednou. Pokud ne, načítá se v jednotlivých iteracích do doby, dokud neobdržíme „***0\r\n\r\n***“. Což je posledních 5 bajtů, symbolizujících konec odpovědi. Pak se z cyklu vystoupí.

ERRORS.CPP

Tento modul obsahuje jednu funkci *errorCall()*, která je volána v případě chyby. Jako parametry přijímá chybovou hlášku a chybový kód, s kterým program ukončí pomocí *exit()*.

Program vrací následující chybové kódy:

- ***-1*** – v případě chyby ve vstupních argumentech programu
- ***-2*** – v případě chyby, která může nastat při zjišťování adresy pomocí *gethostbyname()*.
- ***-3*** – v případě chyb při vytváření a připojování socketu.
- ***-4*** – v případě chyb při vytváření a připojování SSL.
- ***-5*** – v případě ostatních chyb z Discordu, způsobených například: nepřítomností serverů, kde je bot členem, či chybějícím „isa-bot“ kanálem.

INPUT.CPP

V tomto souboru jsou naimplementovány dvě funkce.

První funkcí je *showHelp()*, která v případě vstupního parametru pro nápovědu, tuto nápovědu vypíše na ***STDOUT*** a program se ukončí s návratovou kódem ***0***.

Druhou funkcí je *parseArguments()*. Ta projde veškeré vstupní parametry a provede nad nimi validační kontroly. V případě chybného vstupu je volána funkce *errorCall()* s příslušnou hláškou a chybovým kódem.

PARSE.CPP

Zde nalezneme funkce, zajišťující vytažení potřebných dat z HTTP odpovědi.

První z těchto funkcí je *explode()*, která je obdoba funkce *explode()* z PHP. Rozdělí řetězec podle zadaného oddělovače.

Druhou funkcí je *explodeJson()*, která rozdělí více JSON záznamů podle složených závorek na jednotlivé části a vrátí je ve vectoru.

Další funkcí je *getResponseHeader()*, která oddělí hlavičku od těla a vrátí nám HTTP hlavičku.

Funkcí *getJsonBody()*, je opět rozdělena hlavička a tělo, ale je vráceno HTTP tělo ve formátu JSON.

Funkce *getMessageContent()* je získán obsah zprávy, kterou uživatel zaslal do kanálu „isa-bot“.

Funkce *getMessageAuthor()* se získá buď ID autora zprávy nebo jeho jméno. To závisí od druhého parametru funkce *bool id*. V případě, kdy je potřeba získat ID, v tomto parametru přichází hodnota ***true***.

Funkce *getNewMessages()* rozdělí JSON, ve kterém jsou zprávy, které je potřeba zopakovat. Tento JSON rozdělí na jednotlivé části – zprávy. Ty uloží a vrací ve vectoru.

Funkcí *getRatelimit()* je získáván aktuální limit počtu dotazů, které mohou být zaslány nebo časový limit než dojde k obnovení limitu počtu dotazů. Ten záznam, který je potřeba zjistit, je určen pomocí druhého parametru této funkce.

Funkce *processResponseJson()* zpracovává obecné HTTP odpovědi. Jsou to převážně odpovědi, ze kterých je potřeba získat ID. Jsou to např.: odpověď s client ID, server ID, získání ID kanálu „isa-bot“ s ověřením, jestli tento kanál existuje nebo také získání ID poslední zprávy v kanálu.

Funkce *isResponseChunked()* nám udává, jestli HTTP odpověď přišla jako ***chunked***. To je zjišťováno v HTTP hlavičce.

Poslední funkcí tohoto souboru je *isWrongToken()*, která verifikuje zadaný vstupní bot token. Ověření spočívá v tom, zdali nechodí odpovědi typu „**401-Unauthorized**“.

REQUESTS.CPP

Poslední zdrojový soubor obsahuje funkce pro sestavení dotazů.

Jsou to: *createIDRequest()*, která sestaví dotaz typu ***GET*** pro získání client ID,
createGuildsRequest(), která sestaví dotaz typu ***GET*** pro získání všech serverů, kde je bot členem,
createChannelRequest(), která sestaví dotaz typu ***GET*** pro získání všech kanálů,
createMessagesRequest(), která sestaví dotaz typu ***GET*** pro získání všech zpráv v daném kanálu,

createActualMessagesRequest(), která sestaví dotaz typu ***GET*** s příznakem ***?after=*** k získání všech starších zpráv než zprávy se zadaným *lastMessageID*
a *createSendMessageRequest()*, která sestaví dotaz typu ***POST***, k zaslání zprávy na Discord.

Návod na použití

Aplikaci je možno přeložit pomocí příkazu ***make***, který se vykoná na základě přiloženého souboru *Makefile*. Po tomto překladu se v adresáři vytvoří spustitelný soubor ***isabot***.
Ke smazání přeložených souborů lze použít ***make clean***.
Pro spuštění testů slouží příkaz ***make test***. K těmto testům je nutno mít nainstalované PHP.

Spuštění a ukončení

Program je možno spustit příkazem:
„**./isabot [-h|-help] [-v|--verbose] –t <bot_token>**“
Pořadí parametrů je libovolné. V případě, že je zadán parametr pro nápovědu, je vypsána nápověda a program se dál neprovádí.
Ukončení programu je pomocí CTRL+C (signál SIGINT).

Poznámka

V programu je při vytváření CTX struktury použita ***TLSv1_2_method()***, která je v současné době *deprecated*. Na referenčních serverech Merlin a Eva je starší verze *OpenSSL* a je nutno zde očekávat warnings ohlašující deprecated metodu.

Závěr

Ze zadání se podařilo naimplementovat všechny potřebné části.

Tento projekt mi rozšířil znalosti v oblasti HTTP klient-server. Rovněž mi pomohl k zdokonalení v jazyku C/C++.