Discord bot

ISA

Síťové aplikace a správa sítí

Daniel Kamenický – xkamen21

08. 12. 2019

Obsah

1. Úvod

Cílem projektu bylo naimplementovat komunikující aplikaci se serverem Discord.com. Aplikace působí jako bot na komunikační platformě Discord. Aplikace, po připojení na kanál „#isa-bot“ na Discord serveru, reaguje na všechny příchozí zprávy. Reakce aplikace spočívá v zopakování zaslané zprávy s jménem uživatele, který danou zprávu poslal. Autentizace je prováděna pomocí tokenu, který může být pro každého bota vygenerovaný.

2. Návrh aplikace

Aplikace je členěna do několika modulů. Každý modul se zabývá jinou problematikou a má svůj hlavičkový soubor „.h“. Celá aplikace se překládá pomocí souboru „Makefile“.

2.1 Makefile

Soubor, pomocí kterého se překládá aplikace a definuje závislosti mezi zdrojovými soubory. Soubor vytvoří složku „obj“, do které uloží objektové soubory. Dále vytvoří spustitelný binární soubor „isabot“.

Také slouží pro vyčištění adresáře aplikace do původního stavu. Vymaže vytvořenou složku s objektovými soubory a binární soubor.

2.2 isabot.cpp

Hlavní modul. Slouží pro přehlednost. Probíhá v něm inicializace struktury „Data“ a volání funkce pro kontrolu argumentů (viz. argHandler.cpp).

2.3 argHandler.cpp

Modul „argHandler.cpp“ kontroluje argumenty aplikace. Ukončuje aplikaci při špatném zadání argumentů.

2.4 SSLClient.cpp

Modul „SSLClient.cpp“ obstarává SSL spojení se serverem discord.com a posílání/přijímání paketů. Také kontroluje, zda token aplikace je validní a má autorizační práva k danému kanálu.

Jednou z problematik je doba spojení SSL klienta a serveru. Discord server po nějakém časovém úseku uzavře spojení. To zjistíme pomocí parametru „Connection: close“ v hlavičce odpovědi. V dané situaci modul obstará opětovnou inicializaci a připojení SSL klienta k serveru.

Další problematikou tohoto modulu je souběžně přijímat a odesílat pakety. Modul je tedy spuštěn na dvou vláknech. Každé vlákno se zabývá jednou problematikou. Jedno vlákno pakety přijímá, můžeme ho označit jako „přijímač“, a druhé je zas posílá, můžeme ho označit jako „odesílatel“. Pro vyvarování se souběžného poslání a získávání paketů byl přidán semafor.

Semafor určuje, zda se provádí odesílaní paketu „GET“, nebo je paket přijat a čeká se na jeho zpracování. Odesílatel odešle paket a zablokuje semafor, tím si zamezí možnost odeslání dalšího paketu. Přijímač mezitím obdrží odpověď na zaslaný dotaz. Po zpracování odpovědi odblokuje semafor a odesílatel tak může realizovat odeslání dalšího paketu. Odesílatel také kontroluje, zda se vyskytuje v zásobníku nějaký objekt (viz. stack.cpp). Pokud ano, jedná se o zprávu, kterou má odeslat, a tudíž odešle paket „POST“ s obsahem zprávy. Dotaz „POST“ má vždy přednost před dotazem „GET“. Princip je stejný jak u dotazu „GET“.

2.5 stack.cpp

Modul „stack.cpp“ obstarává práci se zásobníkem. Zásobník realizuje frontu zpráv, kterou aplikace musí odeslat na kanál. Do zásobníku jsou přidávány zprávy od nejnovější, tudíž je pro nás ideální jeho manipulace s daty založena na principu LIFO. Dojde-li k naplnění zásobníku, aplikace vyprázdní zásobník pomocí zasílání dotazů „POST“ (viz. SSLClient.cpp).

```
struct tItem{
    std::string IDOfMessage;
    std::string nameOfAuthor;
    std::string content;
    tItem *next;
};
```

Zpráva je v zásobníku realizována jako struktura „tItem“. Do struktury se ukládá ID zprávy, která slouží pro obnovení poslední ID zprávy (viz. parser.cpp). Jméno autora, kterému patří daná zpráva, a obsah zprávy.

2.6 errorHandler.cpp

Modul „errorHandler.cpp“ obstarává chybové výstupy. Vypisuje chybové zprávy na standartní chybový výstup a ukončuje aplikaci s návratovým kódem.

2.7 parser.cpp

Modul „parser.cpp“ především slouží pro zpracování dat získaných z odpovědí, ale také se zabývá konstruováním dotazů „POST“ a „GET“.

Konstruování dotazů je uděláno sekvenčně. Nejdříve se pošle dotaz „GET“ pro získání ID gildy, ve které je bot. Následuje dotaz „GET“ pro získání všech kanálu daných ID gildy. Poslední dotaz „GET“, který je posílán stále dokola, získává zprávy z kanálu. Pomocí parametru „after“ dotaz vrací pouze zprávy starší, než je zpráva určena zvoleným ID. Modul konstruuje i dotaz „POST“, který má stále stejnou strukturu.

Podle odeslaného dotazu modul vybere, co se má dále provést. Modul zpracovává pouze obsah odpovědi a to tak, že nejdříve odstraní hlavičku odpovědi a poté zpracovává data, které jsou v „JSON“ formátu.

Získání ID gildy a ID kanálu funguje na stejný princip, a proto jsou obě metody implementovány v jedné funkci. Při získávání ID kanálu jsou získány informace o poslední zprávě, a to v podobě jejího ID. Dále je kontrolováno, zdali se jedná o textový kanál. Při získávání jednotlivých zpráv modul prochází data a do zásobníku ukládá informace o jednotlivých zprávách.

```
struct pData{
    std::string botToken;
    std::string IDOfGuild;
    std::string IDOfChannel;
    std::string IDOfLastMessage;
    std::string response;
    bool paramV = false;
    int reset = 0;
};
```

Všechna získaná data jsou ukládána do struktury „pData“. V struktuře se již vyskytuje proměnná „botToken“ a její hodnota byla získána předáním hodnoty parametru „-t“. Slouží pro autentizaci dotazu. Zbylé data jsou získány v průběhu aplikace. Proměnná „IDOfLastMessage“ je měněna podle poslední zaslané zprávy a slouží pro parametr „after“. Samostatné zprávy jsou ukládány do zásobníku.

Modul také kontroluje přehlcení serveru a proto kontroluje „x-ratelimit“ v hlavičce odpovědi a podle toho nastavuje dobu uspání aplikace.

3. Popis implementace

3.1 argHandler.cpp

3.1.1 Funkce get_args

Funkce zpracovává argumenty. Při správném zadání parametrů ukládá jejich hodnotu do struktury „Data“.

```
struct Data{
    bool paramV = false;
    bool paramT = false;
    std::string botToken;
};
```

Modul do struktury „Data“ ukládá informaci o výskytu argumentů a jejich parametrech. Proměnná „paramV“ určuje, zda byl zadán parametr „--verbose“. To stejné dělá proměnná „paramT“, akorát pro parametr „-t“, po kterém musí nadcházet autentizační token příslušného bota. Token je uložen do poslední proměnné „botToken“.

3.2 SSLClient.cpp

3.2.1 Funkce startConnection

Funkce zahájí komunikaci s discord serverem. Inicializuje SSL a pomocné struktury. Při správné inicializaci jsou spuštěny vlákna pro práci s přijímačem a odesílatelem.

3.2.2 Funkce sslInit

Funkce zahájí spojení s SSL klientem. Po inicializaci soketu převede doménové jméno na IP adresu.

```
struct hostent *remoteHost = gethostbyname("discord.com");
struct in_addr in;
bcopy(*remoteHost->h_addr_list++, (char *) &in, sizeof(in));
if(inet_pton(AF_INET, inet_ntoa(in), &serverAddress.sin_addr)<=0){
```

Pro převedení jsou použity knihovny „netdb.h“ a „arpa/inet.h“. Převedení vykonávají funkce „gethostbyname“, „inet_pton“ a „inet_ntoa“. Poté se inicializuje SSL a zahájí se spojení.

3.2.3 Funkce reciever

Funkce je založena na cyklu, z kterého vyskočí v případě špatného čtení nebo přerušení spojení. V cyklu ukládá odpověď do globální proměnné „response“ v struktuře „pData“ a volá funkce pro získání dat z odpovědi. Také kontroluje, zdali byl použit validní token a nepříjde odpověď „401 Unauthorized“.

3.2.4 Funkce SendPacket

Funkce posílá paket s dotazem.

3.2.5 Funkce packetsHandler

Funkce inicializuje potřebné proměnné pro správný chod, jako jsou „request“, nebo „Prequest“ pro ukládání dotazu „POST“ a „GET“, nebo „tmp“ pro získání prvku ze zásobníku. Dále je zahájen nekonečný while cyklus, ve kterém se předávají vytvořené dotazy do funkce „SendPacket“. V cyklu je zanořený druhý nekonečný cyklus, který simuluje odesílání již získaných zpráv. Je přerušen v případě prázdného zásobníku.

3.2.6 Funkce connectionCloseHandler

Funkce hlídá, zda se neukončilo spojení se serverem. Pokud se v odpovědi vyskytlo „Connection: close“, jedná se o ukončené spojení a funkce vrací „1“ pro opětovnou inicializaci SSL klienta.

Získaná odpověď je procházena po znaku. Pro správné nalezení „Connection: close“ je výraz rozdělen na záchytné body. Nejdříve nalezne znak „C“. Pokud mezi dalšími kroky není nalezen znova, hledá se znak „:“. Všechny znaky jsou přidávány do řetězce. Pokud by byl jeden ze znaku nalezen vícekrát, řetězec je vymazán a ukládání znaků začíná od prvního indexu. Po nalezeni „Connection:“ zkontroluje, zdali se jedná od „close“.

3.3 stack.cpp

3.3.1 Funkce initStack

Inicializuje zásobník.

3.3.2 Funkce push

Přidá prvek na vrchol zásobníku.

3.3.3 Funkce pop

Vezme prvek ze zásobníku.

3.3.4 Funkce initItem

Inicializuje strukturu prvek, který je předáván zásobníku.

3.4 errorHandler.cpp

3.4.1 Funkce error

Vypíše chybovou zprávu a ukončí program příkazem „exit“.

3.5 parser.cpp

3.5.1 Funkce getRequest

Funkce vytváří dotazy „GET“. První dotaz je pro získání ID gildy. V druhém dotazu už využívá získané informace a ptá se na kanály v dané gildě. Poslední dotaz, pro získání zpráv, je rozdělen podle výskytu ID poslední zprávy. Pokud se jedná o nový kanál, ve kterém nebyla žádná zpráva, použije se „GET“ pro získání všech zpráv. Pokud se v kanály vyskytují zprávy a aplikace zaznamenala ID poslední zprávy, vypisuje, pomocí parametru „after“, všechny zprávy, které následují za poslední zprávou.

3.5.2 Funkce postRequest

Funkce sestaví dotaz „POST“ se všemi získanými daty z příchozích zpráv.

3.5.3 Funkce doOperation

Funkce určí, zda se se získanou odpovědí provede funkce „guildIDChannelIDLastMessageID“, nebo funkce „parseData“.

3.5.4 Funkce guildIDChannelIDLastMessageID

Funkce obstarává nalezení ID gildy, ID kanálu a ID poslední zprávy. Získaní dat pracuje na stejném principu, a proto je použita jedna funkce. Nejdříve se funkce zbaví hlavičky odpovědi a poté začne procházet JSON data. Celé data jsou procházena postupně a to hledáním podřetězců ohraničených znakem „ “ “. Data jsou porovnávána s patřičnými hodnotami, které jsou potřeba pro získání hodnot.

3.5.5 Funkce parseData

Funkce zpracovává data jednotlivé zprávy. Princip funguje stejně jak u „guildIDChannelIDLastMessageID“, akorát data ukládá do struktury určené pro zprávu a daná struktura je dále uložena do zásobníku. Kontrola zprávy je náročnější z hlediska různorodosti zpráv.

3.5.6 Funkce rateHandler

Vyhledává v hlavičce odpovědi řetězec „x-ratelimit-remaining“, který určuje kolik dotazů zbývá k poslání. Dále vyhledává řetězec „x-ratelimit-reset-after“, který udává počet sekund potřebný pro reset ratelimitu. Při nalezení obou podřetězců a jejich hodnot je porovnána hodnota „x-ratelimit-remaining“ s hodnotou „0“. Pokud souhlasí, tak ve struktuře „pData“ je obnovena hodnota „reset“ hodnotou „x-ratelimit-reset-after“.

4. Informace a Návod na použití

4.1 Spuštění

Spuštění lze provést dvěma způsoby.

4.1.1 Varianta „make“

Příkazem „make“ je aplikace přeložena do spustitelné podoby. Zadáním „./isabot -t <bot_token>“ je aplikace spuštěna.

4.1.2 Varianta „make run“ a „make run_verbose“

Pro variantu „make run“ je zapotřebí upravit soubor „Makefile“ a to přesněji přenastavením hodnoty BOT_TOKEN na hodnotu validního tokenu. Poté zadáním příkazu „make run“ se aplikace přeloží a spustí. Pro spuštění s parametrem „--verbose“ je aplikace spuštěna příkazem „make run_verbose“.

4.2 Omezení

4.2.1 Nastavený discord server

Aplikace předpokládá, že uživatel má vlastní discord účet a vytvořeného vlastního bota, který je přidán na daný server. Server také obsahuje kanál po názvem „#isa-bot“. Při chybějících požadavcích aplikace nepojede. Správné nastavení bota je na stránkách: https://discordpy.readthedocs.io/en/latest/discord.html.

4.2.2 Soubory

Aplikace se umí vypořádat s obrázky či videi a to vrácením jejich proxy url adresa. Při poslání souboru na server se vrátí také proxy url adresa, což pro uživatele je zbytečná informace. Při zaslání souboru je třeba vrátit url adresu, což aplikace nedělá.

4.3 Makefile

4.3.1 Adresář „obj“ a „src"

Zdrojové soubory jsou uložené ve adresáři „src“. Při překladu jsou objektové soubory ukládány do adresáře „obj“. Při spuštění příkazu „make clean“ jsou tyto soubory vyčištěny a je smazána složka „obj“.

5. Chybové výstupy

5.1 return „0“

Správné ukončení programu.

5.2 return „1“

Špatné parametru aplikace.

5.3 return „2“

Špatná inicializace SSL spojení.

5.4 return „3“

Špatné čtení z SSL_reader.

5.5 return „4“

Neautorizovaný token.

5.6 return „5“

ID gildy nebylo získáno z „GET“ dotazu.

5.7 return „6“

Nebyla nalezena gilda.

5.8 return „7“

Nebyl nalezen kanál se jménem „#isa-bot“.

5.9 return „8“

Špatná inicializace zásobníku.

5.10 return „9“

Špatná inicializace prvku zásobníku.

6. Zdroje a Literatura

(1) *O.logN (přezdívka). OpenSSL in C++ Socket Connection (HTTPS Client). 25. 12. 2016* [cit. 13.11.2020]. Dostupné z: https://stackoverflow.com/questions/41229601/openssl-in-c-socket-connection-https-client

(2) *Neznámý autor. Discord Developer Portal [online]. 15.2.2013* [cit. 13.11.2020]. Dostupné z: https://discord.com/developers/docs/intro