VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ISA – Sieťové aplikácie a správa sietí

Projekt: Programovanie sieťovej služby

Varianta: Discord bot

10.11.2020 René Rešetár (xreset00)

Obsah

1 Úvod

Tento dokument popisuje projekt „*Discord bot*“ z predmetu ISA. Jeho obsahom je uvedenie do problematiky, návrh aplikácie, popis implementácie programu, základné informácie o programe a návod na použitie.

Program funguje ako HTTP klient a dokáže zaznamenať, že na Discord kanáli, na ktorom sa nachádza bot sa objavili nové správy. Na tie reaguje správou vo formáte danom v zadaní.

2 Uvedenie do problematiky

Táto kapitola v jednoduchosti vysvetľuje fungovanie HTTP, HTTPS a pojmy s nimi spojené. Taktiež v skratke vysvetľuje pojmy JSON formát, API a Discord API.

2.1 HTTP [1]

Alebo Hypertext Transfer Protocol (Hypertexxtovy prenosný protokol), je protokol definujúci požiadavky a odpovede medzi klientmi a servermi. HTTP klient, alebo *user agent*, ako webový prehliadač zvyčajne započne požiadavku nadviazaním TCP spojenia na určenom porte vzdialeného stroja (80 alebo 443 pre HTTPS). HTTP server, ktorý počúva na danom porte, čaká kým klient nepošle reťazec s požiadavkou (napr. „GET / http/1.1“, ktorý žiada o zaslanie štartovacej stránky webservera) nasledovaný sériou hlavičiek podobných MIME, ktoré popisujú detaily požiadavky a ktoré sú nasledované telesom ľubovoľných údajov (body). Po prijatí požiadavky server pošle reťazec „200 OK“ nasledovaný hlavičkami spolu so samotnou správou, ktorej telo tvorí požadovaný obsah.

Existuje viacero verzií tohto protokolu. Dnes je však najpoužívanejší HTTP/1.1, ktorý požívam aj v projekte popisovanom v tejto dokumentácií.

2.2 HTTPS

Je zabezpečená verzia HTTP. Na ochranu dát používa SSL (Secure Socket Layer)/TLS (Transport Layer Security), kde TLS je novšia, vylepšená a bezpečnejšia verzia SSL. Obe zabezpečujú, že hocijaké prenášané dáta medzi dvoma systémami bude nemožné prečítať. Pri programovaní sa k tomu využívajú SSL/TLS knižnice ako napríklad OpenSSL.

Štandardný port služby je TCP port 443. Je taktiež vhodnejší v prípadoch, kedy je autentifikovaný len jeden koniec spojenia (server). To je typický prípad pri HTTP transakciách cez internet.

2.3 Druhy žiadostí HTTP [2]

Dôležité pre tento projekt:

GET – Zďaleka najbežnejší typ žiadosti. Žiada o zdroj uvedením jeho URL.

POST – Podobne ako GET, okrem toho, že je pridané telo správy.

A ďalšie: **PUT, POST, DELETE, HEAD, TRACE, OPTIONS, CONNECT**

Príklad klientskej požiadavky:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Opera/9.80 (Windows NT 5.1; U; sk) Presto/2.5.29
Version/10.60
Accept-Charset: UTF-8,*
```

(Nasledovaná znakom nového riadku, v tvare ‚\r' nasledovaného znakom ‚\n')

Príklad odpovede servera:

```
HTTP/1.1 200 OK
Content-Length: 3059
Server: GWS/2.0
Date: Sat, 11 Jan 2003 02:44:04 GMT
Content-Type: text/html
Cache-control: private
Set-Cookie:
PREF=ID=73d4aef52e57bae9:TM=1042253044:LM=1042253044:S=SMCc_HRPCQiqyX9j
;
  expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com
Connection: keep-alive
```

(Nasledovaná prázdnym riadkom a zdrojovým textom HTML tvoriacim webstránku Google)

2.4 JSON formát [3]

Namiesto zdrojového textu HTML sme mali za úlohu v tomto projekte spracovávať odpoveď serveru, ktorá bola zaslaná v JSON formáte.

JSON alebo Javascript Object Notation, definuje malé množstvo formátovacích pravidiel na prenosné znázornenie štruktúrovaných dát.

Príklad štruktúrovaných JSON dát kde *"squadName"* a *"homeTown"* sú kľúče pre hodnoty *"Super hero squad"* a *"Metro City"*:

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
}
```

2.5 API a Discord API [4]

API alebo Application programming interface (rozhranie pre programovanie aplikácií) je zbierka funkcií a tried ale aj iných programov, ktoré určujú akým spôsobom sa majú funkcie knižníc volať zo zdrojového kódu programu. API sú programové celky, ktoré programátor volá namiesto vlastného naprogramovanie.

Discord ponúka otvorené rozhranie API slúžiace na žiadosti o integrácie robotov aj OAuth2. Discord API je založené na dvoch základných vrstvách, HTTPS/REST API pre všeobecné operácie a trvale zabezpečené pripojenie WebSocket na odosielanie a prihlásenie na odber v reálnom čase. V tomto projekte si vystačíme s HTTPS/REST API.

3 Návrh riešenia

Prvotne bolo v tomto programe potrebné spracovať argumenty pre chcenú funkcionalitu programu. Nasledovne bolo nutné nadviazať TCP spojenie s komunikačnou službou Discord, kde bolo potrebné pripojiť sa na Discord server na kanál „#isa-bot“ pomocou overovacieho tokenu bota, ktorý je na tomto kanáli pridaný. Potom program musí v opakujúcom sa intervale zasielať žiadosti pomocou Discord API. Ich pomocou zisťuje, či sa na kanáli objavili nejaké nové správy. Ak áno je potrebné tieto správy spracovať (zistiť odosielateľa a obsah jednotlivých správ). Podľa odosielateľa program rozhodne, či na danú správu bude reagovať.

4 Implementácia

Popisuje ako som pri riešení projektu postupoval a aké prostriedky som použil a ako program funguje. Pri implementácií tejto sieťovej aplikácií som sa rozhodol pre jazyk C++ prekladaný prekladačom *g++* pomocou súboru *Makefile*. Vybral som si spoľahlivosť odpovede pred rýchlosťou, takže aj keď to niekedy trvá trocha dlhšie, program by mal vždy správne reagovať.

4.1 Parsovanie argumentov

Argumenty som spracovával pomocou *getopt.h*. Vytvoril som si pomocnú štruktúru pre *long_options*. Vo *while* cykle pomocou switchu koordinoval čo má program vykonať popri objavení argumentu. Celá kontrola sa nachádza v *main()* funkcii. Ak boli argumenty správne program pokračuje funkciou *bot(c)*, kde c je mnou vytvorená štruktúra *Connection*, v ktorej prenášam potrebné informácie.

4.2 Nadviazanie spojenia

Na začiatku si vytvorím TCP socket pre adresu *„discord.com“* na porte *443* a pre *IPv4 addres family*. Nasledovne inicializujem SSL metódu pomocou:

```
const SSL_METHOD *meth = TLSv1_2_client_method();
SSL_CTX *ctx = SSL_CTX_new (meth);
ssl = SSL_new (ctx);
```

Potom pre SSL objekt získame súborový deskriptor , ktorý napojíme na SSL objekt.

```
sock = SSL_get_fd(ssl);
SSL_set_fd(ssl, s);
```

Popri tom program samozrejme kontroluje zlyhania inicializácie a v ich prípade sa ukončí s príslušným výpisom a vráti 1. Inak pokračuje kontrolou tokenu *check_token(string token)* a získaním potrebných údajov (*guild_id, channel_id, last_message_id*) pomocou *get_basic_info(struct Connection)*.

4.3 Posielanie správ

Správy program zasiela pomocou *SendPacket(char* message)* [1], kde *message* argument predstavuje *žiadosť* ktorý chceme zaslať. Funkcia volá *SSL_write(SSL*, const void*,int)* a kontroluje či nedošlo k chybe.

4.4 Prijímanie správ

Správy program prijíma pomocou upravenej *RecvPacket()* [1]. Funkcia v cykle while pripraví buffer pomocou *memset()* a číta odpoveď po častiach o veľkosti *sizeof(buffer)*, do ktorého ju dočasne uloží. V tejto časti prvotne kontroluje aký máme limit na žiadosti a to pomocou *check_if_sleep(string msgs)*, ktoré nájde *„x-ratelimit-remaining“* v hlavičke odpovedi. Ak je jeho hodnota ‚0‘ skontroulujm hodnotu *„x-ratelimit-reset-atfter“* a uspím na jeho hodnotu. Potom odpoveď kontrolujem na chybové správy:

```
if(regex_search(ch, regex("(.)"message": "404: Not Found", "code": 0(.)")))
{
    error("Not Found 404\n");
}
if(regex_search(ch, regex("(.)403 Forbidden(.)"))) {
    error("Forbidden 403\n");
}
```

Ak prešiel kontrolou dopíše túto časť odpovede do *result* a skontroluje chyby spojenia. Cyklus beží kým nedôjde na koniec priatej odpovede:

```
} while (!regex_search(chunk, regex("(.*)0\r\n\r\n")));
```

Ak všetko prebehne bez chyby vráti celú odpoveď. Ak nie tak vráti prázdny reťazec alebo ukončí program s chybovým hlásením a vráti 1 (pomocou *error(char* msg)* ktorá vracia *int*).

4.5 JSON parsovanie

Pre parsovanie odpovede a *JSON body* z odpovede som vytvoril almost_json_pars.h a almost_json_pars.cpp kde som naprogramoval pomocné funkcie na spracovanie odpovedí servera. Využil som *string::find()* a prechádzanie textu v cykle *for*.

4.6 Spracovanie nových správ

Hlavná funkcionalita bota prebieha v nekonečnom cykle *while(true)* po ubehnutí opakujúceho sa časového intervalu zasiela žiadosť na zistenie či sa na kanáli objavili nové správy:

```
msg = "GET /api/v6/channels"/;
msg.append(con.channel_id);
msg.append("/messages?after=");
msg.append(con.last_message_id)
```

Získa odpoveď a tú skontroluje. Ak sa tam našli nové správy, obnoví hodnotu *con.last_message_id* a volá *handle_messages(string msgs, Connection c).*

Tam pomocou *get_msg_info(string msgs)* uloží všetky potrebné informácie o správach, ktoré nie sú od botov, do vektorov. Tieto vektory posunie ako jeden z argumentov do *send_msgs(Vectors vectors, Connection co)*. Vďaka vektorom môže správy s ľahkosťou skladať a zasielať v správnom poradí.

Taktiež ak bola v argumentoch prítomná možnosť **[–v|--verbos]** vypisujem na *stdout*.

4.7 Testovanie

Program som testoval som vytvorením dvoch ďalších botov „*Botanik*“ a „*testbotik*“. Pomocou ich tokenov som cez *Postman* aplikáciu zasielal správy ako je možné vidieť na obrázku nižšie.

Taktiež som testoval limity pridaním ľudí na server a zasielaním správ na kanál v rôznych formátoch, intervaloch a o rôznych veľkostiach (kde som sa pozeral hlavne na poradie odpovedí a ako si poradí s veľkým množstvom správ za krátky čas).

Do *make test* som zaradil iba testy kontrolujúce chybové argumenty.

5 Použitie

Program sa spúšťa príkazom *./isabot* za pomoci prepínačov:

-t <token> [-h|--help] [-v|--verbose]

Program je pred spustením potrebné skompilovať:

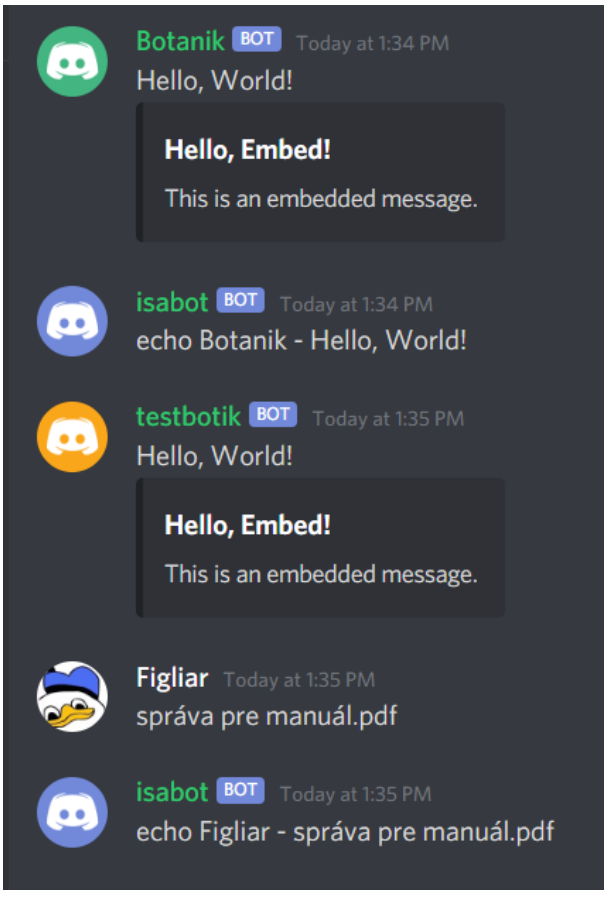
```
$ make clean
$ make
```

Program sa používa nasledovne:

```
Pre nápovedu:
$ ./isabot
$ ./isabot [-h|--help]
Pre echo na Discord kanáli:
$ ./isabot -t <token>
Pre echo na Discord kanáli aj na stdout:
$ ./isabot -t <token> [-v|--verbose]
$ ./isabot [-v|--verbose] -t <token>
Na poradí argumentov nezáleží. Iba za -t musí nasledovať token.
```

Ukážkový výstup:

```
$ ./isabot -t 45684513sdasaf45as.mojvymyslenytoken.15132132ds1231d -v
isa-bot - Botanik: Hello, World!
isa-bot - Figlair: spr\u00e1va pre manu\u00e1l.pdf
```



Obrázok 1. Test odpovede programu podľa mena používateľa

Pre spustenie testov správnosti argumentov (podmienkou je prítomnosť bashu, čiže na merlinovi pôjde ale na eve nie):

```
$ make test
```

5.1 Obmedzenia

Keďže je program obmedzený hodnotou *x-ratelimit-remaining* tak mu niektoré veci môžu trvať (keď som ho však obmedzoval inak, nie vždy správne fungoval pri väčšom množstve správ a na niektoré neodoslal odpoveď na Discord kanál). Program na *stdout* nevypisuje Unicode znaky. Program berie aj dlhšie argumenty ako --verb, --ver, --verbos... (všetko medzi --v a --verbose). Argumenty ako --verbose4 alebo -vsp už neberie.

5.2 Chybové správy

Pri chybe počas čítania odpovede serveru:

```
if (len < 0) {
    int err = SSL_get_error(ssl, len);
    if (err == SSL_ERROR_WANT_READ)
        error("err == SSL_ERROR_WANT_READ");
    if (err == SSL_ERROR_WANT_WRITE)
        error("err == SSL_ERROR_WANT_WRITE");
    if (err == SSL_ERROR_ZERO_RETURN  err == SSL_ERROR_SYSCALL  err ==
SSL_ERROR_SSL)
    error("err == SSL_ERROR_ZERO_RETURN  err == SSL_ERROR_SYSCALL  err ==
SSL_ERROR_SSL");
}
```

Pri kontrole argumentov (Ak užívateľ zadá zlé argumenty):

```
error("Error, wrong arguments!\n");
```

Pri chybe počas získavania základných informácií (Ak program dostane prázdnu odpoveď):

```
error("Error getting guilds\n");
error("Error getting channels\n");
error("Error getting messages\n");
```

Ak užívateľ zadá chybný token:

```
if(token.length() != 59) error("Error, wrong token\n");
error("Error, getting info about @me\n");
```

Pri chybe počas vytvárania spojenia (Ak sa nepodarí nejaká časť inicializácie spojenia):

```
if (s < 0)  error("Error creating socket.\n");
if (connect(s, (struct sockaddr *)&sa, socklen)) error("Error connecting to
server.\n");
if(ctx == nullptr) error("Error calling SSL_CTX_new()\n");
if(!ssl){
    fprintf(stderr, "Error creating SSL.\n");
if(err <= 0) {
fprintf(stderr, "Error creating SSL connection.  err=%x\n", err);
```

6 Záver

Prácu na tomto projekte hodnotím pozitívne. Zadanie bolo zaujímavé, informácie k štúdiu a implementácií sa dali nájsť pomerne ľahko a pri programovaní tohto projektu som sa aj niečo naučil. Moje riešenie určite nie je dokonalé ale malo by spĺňať všetky zadané požiadavky a obmedzenia.

Program som otestoval na svojom počítači (Ubuntu) a na školských serveroch *merlin* a *eva*. Pri mojom testovaní fungoval projekt ako bolo mienené. Za prípadné implementačné alebo iné chyby sa vopred ospravedlňujem.

7 Referencie

[1] https://tools.ietf.org/html/rfc7235

[2] https://tools.ietf.org/html/rfc7231

[3] https://tools.ietf.org/html/rfc8259

[4] https://discord.com/developers/docs/intro

[5] https://gist.github.com/AhnMo/d288652b13cec77bf89b39186d07bf28