Vysoké učení technické v Brně
Fakulta informačních technologií

Počítačové komunikace a sítě – 1. projekt
Klient-server pro získání informace o uživatelích

Varianta *I*

2018 Tomáš Zubrik

Obsah

1 Úvod

Schopnosť posielať dáta po sieti medzi viacerými počítačmi je potrebná a nevyhnutná pre vývoj sieťových aplikácií. Dokumentácia k projektu popisuje návrh aplikačného protokolu a implementáciu klientskej a serverovej aplikácie v jazyku C realizujúcu spostredkovanie informácií o užívateľoch na serveri. Byť schopný posielať relevantné a spoľahlivé správy medzi počítačmi je dôležité a metóda ktorú popíšeme je znovupoužiteľná aj v iných aplikáciach.

2 Použitie TCP

2.1 TCP Protokol

Na transportnej vrstve je použitý protokol TCP. Navrhnutý aplikačný protokol je spoľahlivý, pretože TCP je spojovo orientovaný protokol pre prenos bajtov so spoľahlivým doručovaním. Pred odosielaním dát sa musí naviazať spojenie medzi klientom a serverom k čomu slúži trojcestný handshaking (three-way handshake).

2.2 Konfigurácia serveru

Prvý krok konfigurácie serveru je nastavenie soketu na špecifickom porte, druhý krok je aktívne čakať na požiadavky od klientov. V implementácii to má na starosti funckia `start_server()`, ktorá otvorí soket na danom porte, nastaví príznak znovupoužiteľnosti adresy, zviaže soket s danou adresou a následne čaká na klientské požiadavky. Spojenie musí byť potvrdené a prijaté funkciou `accept_connection()`, ktorá vracia číslo reprezentujúce komunikačný soket klienta a servera.

2.3 Konfigurácia klienta

Konfigurácia klienta je podstatne jednoduchšia než konfigurácia serveru. Kľúčová funkcia na pripojenie klienta na server je `connect_to_server()`, v ktorej pripojenie na server je možné použiť buď jeho IP adresu alebo DNS meno.

2.4 Zasielanie a prijímanie správ

Správy zasiela ako klient tak i server. Vstupné argumenty pri spustení klienta musia byť doručené na server, kde sa vyhodnotí aké informácie o užívateľoch má server odosielať naspäť klientovi. Vhodne implementované funkcie na odosielanie správ sú `send_data_message()`, ktorá posiela spolu s typom správy aj užitočný obsah tzv. payload a `send_ack_message()`, ktorá neodosiela užitočné dáta, dôležitý je typ správy vďaka ktorému sme schopní sledovať tok správ medzi klientom a serverom.[2]

3 Dizajn aplikačného protokolu a využité štruktúry

3.1 Štruktúra String

Implementácia štruktúry String viedla k jednoduchej správe pamäti a prípadnej realokácii, ak dĺžka informácie presiahla alokované miesto. Štruktúra String bola využitá ako na strane servera - uložené informácie option a login zaslané od klienta, tak na strane klienta, ktorému boli zaslané relevantné informácie na základe jeho požiadavky a tie boli uložené do Stringu na strane klienta.

3.2 Štruktúra Socket

Štruktúra Socket uchováva takmer všetky dôležité informácie pre prepojenie klienta a servera ako file descriptor soketu, štruktúru adresy a dĺžku adresy a pod.

3.3 Štruktúra Message

Správa, ktorú posiela buď server alebo klient obsahuje dva typy informácií. Prvý je typ posielanej správy `message_type` a druhý sú užitočné dáta tzv. `payload` o veľkosti makra `PAYLOAD_SIZE`. Inšpirácia z odkazu.[1]

```
typedef struct
{
        enum message_type type;
        char payload[PAYLOAD_SIZE];
} Message;
```

3.3.1 Typy správ

Definovanie typov správ je fundamentálna časť dizajnu aplikačného protokolu. Pre jednoduchosť je typ správy definovaný vymenovaním všetkých typov pomocou enum. Každý druh správy má svoje príslušiace číslo, ktorým sa v danej množine identifikuje.

```
enum message_type{CONNECT_REQ = 1, CONNECT_OK = 2, CONNECT_FAIL = 3,
    OPTION_REQ = 4, OPTION_SENT = 6, OPTION_OK = 7, ... SUCCESS = 20,
    FAILURE = 21, EXT_FAILURE = 22, EXT_SUCCESS = 23};
```

3.3.2 Payload

Užitočné dáta tzv. payload reprezentujú typ char*. Obsahuje dáta, ktoré si medzi sebou posielajú klient a server. Veľkosť je odhadnutá vzhľadom na požiadavky klienta. Nadmerne veľký payload zvyšuje pamäťové nároky, veľmi malý zvyšuje réžiu pri posielaní správ po sieti.

```
#define PAYLOAD_SIZE 256
```

4 Implementácia a demonštrácia

4.1 Zaujímavé časti

4.1.1 Funkcia getpwnam()

Pre získanie informácií o užívateľoch na strane serveru sa vhodne využila funkcia `getpwnam()` z hlavičkového súboru `pwd.h`, ktorá má ako vstupný argument login užívateľa a vracia ukazateľ na štruktúru `struct passwd`, ktorá obsahuje osobné informácie, informácie o priečinku a pod.

```
struct passwd *user_info = getpwnam(login);
```

4.1.2 Funkcia print_info()

Pre vypisovanie zaslaných a prijatých správ slúži funkcia `print_info`. Je zakomentovaná z dôvodu splnenia zadania, ale pre pochopenie komunikácie je veľmi jednoduchá. Informuje, či správa bola odoslaná alebo prijatá a akého je typu.

```
print_info("Message received", "SUCCESS");
```

4.1.3 Acknowledge funkcie

Funkcie `send_ack_message` a `send_ack_message` slúžia na zasielanie tzv. *acknowledgment* správ. Správy, ktoré potvrdia správanie, aké očakáva druhá strana komunikácie. Neposielajú si dáta v payloade, posiela sa iba typ správy, podľa ktorého sa vyhodnotí správnosť a celá logika komunikácie.

```
/* Prototypes */
void send_ack_message(int socket_fd, Message *msg, int msgtype);
int recv_ack_message(int socket_fd, char *buffer, Message *msg, int
    buffer_size);

/* Usage */
send_ack_message(socket_fd, &outgoing_message, SUCCESS);
recv_ack_message(socket_fd, (char *)&incoming_message, &incoming_message,
     sizeof(incoming_message));
```

4.1.4 Posielanie dát

Na posielanie dát sú využité funkcie `send_data_message` a `send_data_message`, ktoré v cykle prijímajú a odosielajú správy medzi klientom a serverom. Používajú payload[PAYLOAD_SIZE] a efektívne pracujú so štruktúrou String do ktorej konkatenujú reťazec prijatý v správe.

```
/* Prototypes */
void send_data_message(int socket_fd, Message *msg, char *data);
int recv_data_message(int socket_fd, Message *msg, String *data);

/* Usage */
send_data_message(socket_fd, &outgoing_message, data);
recv_data_message(socket_fd, &incoming_message, list);
```

4.2 Demonštrácia činnosti aplikácie

./ipk-client -h merlin.fit.vutbr.cz -p 55551 -l z
zachariasova
zahorik
zavadila
zboril
zborilf
zeinali
zemcik
zendulka
zezula
zizkaj

./ipk-client -h merlin.fit.vutbr.cz -p 55551 -f error
ERR ... Username you set doesn't exist! Run program with [-l] option to see all usernames.

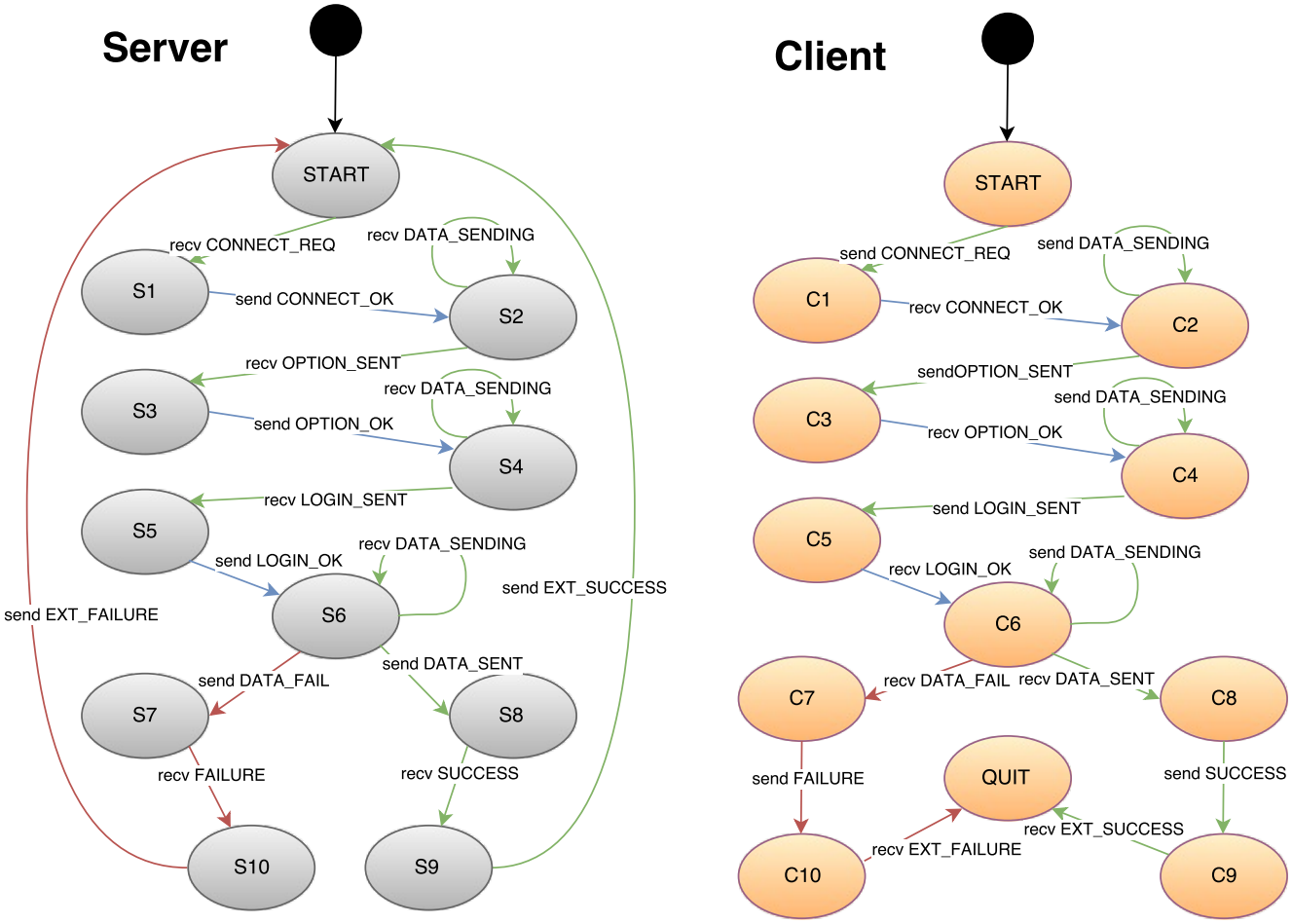
./ipk-client -h merlin.fit.vutbr.cz -p 55551 -f xzubri00
/homes/eva/xz/xzubri00

./ipk-client -h merlin.fit.vutbr.cz -p 55551 -n xzubri00
/Zubrik Tomas,FIT BIT 2r

4.2.1 Demonštrácia s výpisom info správ

./ipk-client -h merlin.fit.vutbr.cz -p 55551 -n xzubri00
INFO ... Message sent CONNECT_REQ
INFO ... Message received CONNECT_OK
INFO ... Message sent DATA_SENDING
INFO ... Message sent OPTION_SENT
INFO ... Message received OPTION_OK
INFO ... Message sent DATA_SENDING
INFO ... Message sent LOGIN_SENT
INFO ... Message received LOGIN_OK
INFO ... Message received DATA_SENDING
INFO ... Message received DATA_SENT
Zubrik Tomas,FIT BIT 2r
INFO ... Message sent SUCCESS
INFO ... Message received EXT_SUCCESS

5 Aplikačný protokol FSM



6 Referencie

[1] Lattrel, R. *Designing and Implementing an Application* [online]. 2012 [cit. 2015-12-13]. Dostupné na: https://www.egr.msu.edu/classes/ece480/capstone/fall12/group02/documents/Ryan-Lattrel_App-Note.pdf

[2] [online]. Dostupné na:http://www.faqs.org/docs/artu/ch05s03.html