VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 3. projekt

Chatovací klient

23. dubna 2017 Jakub Frýz

Úvod

Tato dokumentace se věnuje procesu (co dělá) chatovacího klienta jakožto projektu do IPK.

Projekt byl primárně vyvíjen na **Kubuntu 17.04** Zesty Zapus. Byl i otestován na referenčním přístroji **CentOS7**.

V následujících sekcí jsou jednotlivě popsány základní části programu. Většina částí má ošetřené chyby výpisem chybové zprávy a ukončením.

1 Argumenty

Pravděpodobně nejlehčí část projektu. Program očekává 4 argumenty, ale kontroluje pouze první a třetí. Pro ulehčení práce jsou argumenty převedy na typ `string`. Není-li jeden z nich `-i` a druhý z nich `-u`, program se ukončí s chybou. Jsou-li duplicitní, program se též ukončí chybou. Uživatelské jméno se uloží do proměnné `username` typu `string` a IP se ulože pouze index v argumentech (index `i+2` z důvodu, že funkce `inet_pton()` nepřijme řetězec typu `string`, ale pouze typu `char*`, a proto se ukládá index na původní argumenty, které jsou navíc o jedno posunuty).

2 Připojení k serveru

Nejdříve se provede vytvoření socketu (funkce `socket()`) a nacpou se potřebné informace do struktury `server_addr` jako port a IP adresa. Zde se zároveň i zkontroluje správnost IP adresy (funkce `inet_pton()`).

Pak už nezbývá nic jiného než se připojit na server (funkce `connect()`).

3 Tvorba vlákna

Pro správnou funkcionalitu klienta je potřeba obsluhovat `stdin` a `stdout` zároveň. Tento problém jsem se rozhodl vyřešit pomocí POSIX vlákna (`pthread`). Na vytvořeném (funkce `pthread_create()`) vlákně mi pak jede funkce `void *chat_recv(void *threadarg)`, která je podrobněji popsána v sekci 5. Vlákno se před ohlášením a po přijmutí signálu `SIGINT` ukončí (funkce `pthread_cancel()` a `pthread_join()`).

4 Odesílání zpráv

Odesílání zpráv jsem zanechal v hlavním procesu programu. Zprávy se odesílají pomocí funkce `send()`. Prvně odešle zpráva, že se uživatel přihlásil.

Poté se program zacyklí a při každém cyklu kontroluje, zda-li nebyl přijmut signál `SIGINT`. Zde je další cyklus, který obsluhuje neblokující čtení ze `stdin` za pomoci funkce `kbhit()`. Funkce `kbhit()` ve své podstatě přidává časový limit pro čtení ze vstupu. Po zmačknutí `ENTER` může funkce `getline()` načíst zprávu do proměnné `line` typu `string`. Zkontroluje se, že zpráva není prázdná. Následuje spojení zprávy. To se provede do proměnné `msg`, do které se postupně přiřadí uživatelské jméno, dvojtečka s mezerou, zpráva samotná a zalomení řádku (`\r\n`). Výsledná zpráva se musí ještě před odesláním převést na typ `char*` (funkce `convert()`) a pak se může odeslat.

Po přijmutí signálu `SIGINT` se cyklus ukončí a odešle se odhlašovací zpráva.

5 Přijímání zpráv

Zprávy se přijímají ve vlákně a víceméně to funguje stejně odesílání zpráv. Je zde cyklus, který čeká na příchozí zprávy (funkce `recv()`). Nejdříve se však vyčistí `buffer`, do kterého se zpráva uloží. Potom se zpráva bez jakýchkoliv úprav pošle přímo na standardní výstup `stdout`.

6 Signál `SIGINT`

Pro obsluhu signálu `SIGINT` je zde funkce `sigint_handler()`. Ta je určena pomocí funkce `signal()` už na začátku programu. V případě, že program příjme signál SIGINT, je funkce `sigint_handler()` vyvolána a pozmění hodnotu globální proměnné `stop`, což zapříčiní ukončení jinak nekonečných cyklů pro odesílání a přijímání zpráv.

Závěr

Přiznám se, že jsem měl z tohoto projektu trochu strach, ale nakonec jsem se bavil. Spoustu jsem se toho přiučil. Před tímhle projektem jsem vůbec neměl ani ponětí o obsluze signálů či neblokujícím čtení ze vstupu. Myslel jsem si, že zprovoznění chatovacího klienta bude trošku obsáhlejší, co se do počtu řádků kódu a použitých příkazů, ale asi jsem se mýlil. Sice nemůžu konkurovat *Facebooku* nebo *Skypu*, ale rozhodně se můžu pochlubit tím, že jsem si vytvořil chatovacího klienta (s pár „mouchama“). Teď už by to chtělo jen grafickou nástavbu.