VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 1. projekt

Varianta 1
Klient-server pro získání informace o uživatelích

# Dokumentace

12. března 2018

Petr Marek

# Obsah

# 1 Zadání

Cílem projektu bylo vypracovat aplikaci zajišťující spolehlivou komunikaci mezi serverem a klietem. Projekt se dělí na dvě hlavní části. První je návrh aplikačního protokolu, který zajišťuje spolehlivý přenos informací. Druhá část je naprogramování samotné aplikace v C/C++. Ta realizuje zprostředkování dat o uživatelích na straně serveru, ze souboru */etc/passwd*.

# 2 Výcuc relevantních informací

Zadání říká, že přenos dat má být spolehlivý. Na základě informací z RFC 793[4], jsem se rozhodl využít protokolu TCP, který toto umožňuje. Data jsou v něm přenášena spojitě po tocích bytů. TCP má přehled o tom, které byty již byly odeslány a které na odeslání teprve čekají. Navíc pokud dojde ke špatnému pořadí příjmu dat, TCP je nejprve správně seřadí a teprve poté je označí za přijmuté.

K vypracování projektu jsem se inspiroval systémem zpráv protokolu SMTP[3]. Ovšem nic takové jako zpráva na úrovni TCP neexistuje. Pokud chceme odesílat zprávy, je nutné rozlišit konec jedné a začátek druhé. K tomu slouží dva přístupy. První je prefixování délkou, druhý je použití oddělovače[1].

# 3 Aplikační protokol

Předtím, než může započít komunikace mezi serverem a klietem, je nutné, aby byl server již aktivní. Komunikace je iniciována klientem. Ten odesílá serveru zprávy o tom, o která data ze souboru server žádá. Součástí každé zprávy je i její kódové označení. Na jeho základě obě strany rozpoznají typ zprávy, kterou právě obdrželi[3]. K rozlišení zpráv jsem použil prefixování délkou. Před samotnými daty se odešlou čtyři bajty udávající jejich velikost v bytech. Poslední částí zprávy jsou tři bajty udávající její kódové označení. Podle tohoto kódu rozpoznáme, o jaký typ se jedná.

Na následujících řádcích popíši způsob komunikace serveru s klientem za předpokladu, že vše probíhá bez chyb. Nejrpve pošle klient serveru zprávu s kódovým označením FLAG. Server po jejím příjmu odpovídá kódem FLAG_OK. Klient přijímá FLAG_OK a odesílá LOGIN. Server přijímá LOGIN a odesílá LOGIN_OK. Následně server odesílá zprávu s kódem MSG_CNT, ve které je informace o velikosti dat, které bude klient přijímat. Klient na to odpovídá MSG_CNT_OK. Následně server odešle data klientovi s kódovým označením DATA. Ten je přijímá a odesílá DATA_OK.

Pokud nastane jakákoliv chyba, strana na které se chyba vyskytla odešle druhé straně zprávu s kódovým označením FAILURE. Součástí zprávy jsou i data s popisem chyby.

# 4 Implementace

Úmyslně jsem při implementaci rámcování zpráv nevyužil strukturu pro zaobalení dat[2]. Tam by nastal zbytečný problém s určením velikosti takové zprávy a kompatibilitě velikostí typů. Namísto toho odesílám data postupně část za částí jako byty a tak je i poté čtu. Tento přístup nejspíš nebyl jednodušší, ale je transparentnější.

Samotná data odesílám po vybraných řádcích souboru. Tento způsob je pomalejší, pokud je dat velké množství. Na druhou stranu není nutné velké objemy dat rozdělovat na menší, které by se optimálně vešli do předem specifikované velikosti paměti (1024 bytů). Nicméně uznávám, že zde by byl prostor pro optimalizaci.

Chybové stavy jsou vyřešeny následovně. Pokud chyba nastane na straně klienta, informuje o tom server. Poté vypíše chybové hlášení na standardní chybový výstup (stderr) a skončí s chybou. Pokud je chyba na straně serveru, informuje o ní klienta a pouze vypíše chybu na stderr. Nekončí s chybou proto, že na serveru mohou být připojeni další klienti, kteří touto chybou nemusí být ovlivněni. Činnost serveru je tedy, mimo jeho nesprávné spuštění, ukončena výhradně signálem SIGINT.

Pro implementaci serveru, schopného obsloužit více klientů, jsem použil fork. Ten vytvoří pro každého klienta zvláštní proces, ve kterém bude obsloužen. Jelikož se v těchto dílčích procesech pracuje se stejným souborem, bylo třeba také implementovat mutex (semafor), který zajišťuje vzájemné vyloučení těchto procesů.

# 5 Příklady použití

```
./ipk-server -p 4000

./ipk-client -h localhost -p 4000 -l xmarek
xmarek02
xmarek59
xmarek62
xmarek64
xmarek66
xmarek67
xmarek69

./ipk-client -h localhost -p 4000 -l xmarek66
xmarek66

./ipk-client -h localhost -p 4000 -f xmarek66
/homes/eva/xm/xmarek66

./ipk-client -h localhost -p 4000 -n xmarek66
Marek Petr,FIT BIT 2r
```

# Reference


[1] CLEARY, S. *Message Framing* [online]. Poslední změna 30. dubna 2009 [cit. 10. dubna 2018]. Dostupné na: <https://blog.stephencleary.com/2009/04/message-framing.html>.

[2] LATTREL, R. *Designing and Implementing an Application Layer Network Protocol Using UNIX Sockets and TCP* [online]. [cit. 10. dubna 2018]. Dostupné na: <https://www.egr.msu.edu/classes/ece480/capstone/fall12/group02/documents/Ryan-Lattrel_App-Note.pdf>.

[3] RAYMOND, E. S. *The Art of Unix Programming* [online]. [cit. 10. dubna 2018]. Dostupné na: <http://www.faqs.org/docs/artu/ch05s03.html>.

[4] REY, C. . Information Sciences Institute University of Southern California 4676 Admiralty Way Marina del. *TRANSMISSION CONTROL PROTOCOL* [online]. Poslední změna září 1981 [cit. 10. dubna 2018]. Dostupné na: <http://www.freesoft.org/CIE/RFC/793/>.