VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Dokumentácia k projektu do predmetu IPK

Klient-server aplikácia pre získanie informácie o užívateľoch

12. marca 2018

Matúš Liščinský (xlisci02)

Obsah

1 Úvod

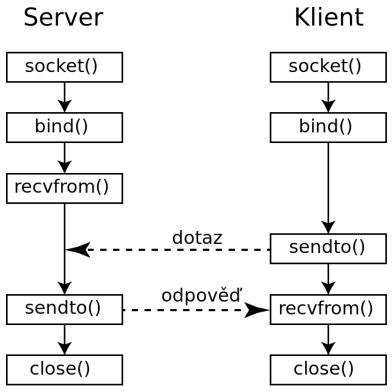
Dokumentácia popisuje implementáciu klientskej a serverovej aplikácie realizujúcej prenos informácii o užívateľoch na strane serveru na základe modelu Klient-server za pomoci vlastného aplikačného protokolu. Sprostredkúvané informácie o užívateľoch server získáva zo súboru `/etc/passwd`.

2 Model Klient-server

Model klient-server rozlišuje systémy klienta od systémov servera, ktoré komunikujú cez počítačovú sieť. Klient-server opisuje vzťah medzi dvoma počítačovými programami, z ktorých jeden, klient, odošle požiadavku na službu z druhého programu, servera, ktorá požiadavku splní. Aj keď tento model môže byť použitý programami na jednom PC, väčšie uplatnenie nachádza v sieti. V sieti je tento model vyhovujúcim spôsobom na efektívne prepojenie programov, ktoré sú distribuované po rôznych miestach v sieti. Väčšina internetových aplikácií, ako email, prístup na web a k databázam, sú založené na tomto modeli. [3]

3 Berkeley sockets (BSD Sockets)

Rozhranie BSD Sockets poskytuje užívateľským aplikáciám jednotný prístup k transportnej a prípadne nižším vrstvám protokolového stacku. Rozhranie Sockets je navrhnuté tak, aby sa javilo ako rozšírenie Unixového súborového systému. Týmto je možné využívať sady funkcií určených pre prácu so súbormi aj pre komunikáciu prostredníctvom siete. Pretože je sieťová komunikácia značne obsiahlejšia čo do množstva neštandardných stavov, ktoré môžu nastať, obsahuje Sockets aj množstvo prídavných funkcií, ktoré niesú v súvislosti s bežnými súborovými operáciami potrebné.



Obr. 1: Príklad zostavenia spojenia s využitím BSD socketov.

Pre umožnenie implementácie rozličných skupín protokolov definuje rozhranie Sockets pojem *rodiny protokolov* (protocol family). Rodina protokolov zahŕňa skupinu protokolov používaných v istom komunikačnom prostredí - napríklad rodina PF_INET pre použitie v Internete zahŕňa protokoly IP, TCP a UDP.

Pod pojmom *socket* rozumieme koncový komunikačný uzol spojenia. Z hľadiska softvéru sa vlastne jedná o dátovú štruktúru, popisujúcu stav, v ktorom sa komunikujúci uzol nachádza. To zahŕňa okrem iného adresu lokálneho (a príp. vzdialeného) účastníka spojenia a u virtuálneho kanálu taktiež jeho stavovú informáciu. Pojem socket sa objavuje aj v špecifikáciach protokolov TCP a UDP, kde je definovaný ako dvojica tvorená **IP adresou** a **číslom portu**, jednoznačne identifikujúca proces na príslušnej stanici. Spojenie je potom jednoznačne určené dvojicou takýchto socketov, niekedy označovanú ako *socket pair*. [1]

4 Návrh aplikačného protokolu

Táto sekcia obsahuje súhrn pravidiel, ktoré zabezpečujú komunikáciu medzi programom klienta a programom servera na najvyššej, aplikačnej vrstve. Pomocou protokolu TCP si tieto 2 strany vymieňajú dáta. Dáta sú v tomto prípade zapuzdrené do štruktúry obsahujúcej typ správy a užitočné dáta. [2]

4.1 Detail implementácie

```
#define BUFSIZE 256

/* typ spravy */
enum msg_type{ DATA, DATA_END, SERVER_ERR, LIST, HOME_DIR, USER_INFO,
CONFIRM_END, CONNECT_REQ, CONNECT_OK, CONNECT_FAIL};

/* struktura reprezentujuca spravu */
typedef struct {
    enum msg_type flag;
    char data[BUFSIZE];
}Message_t;
```

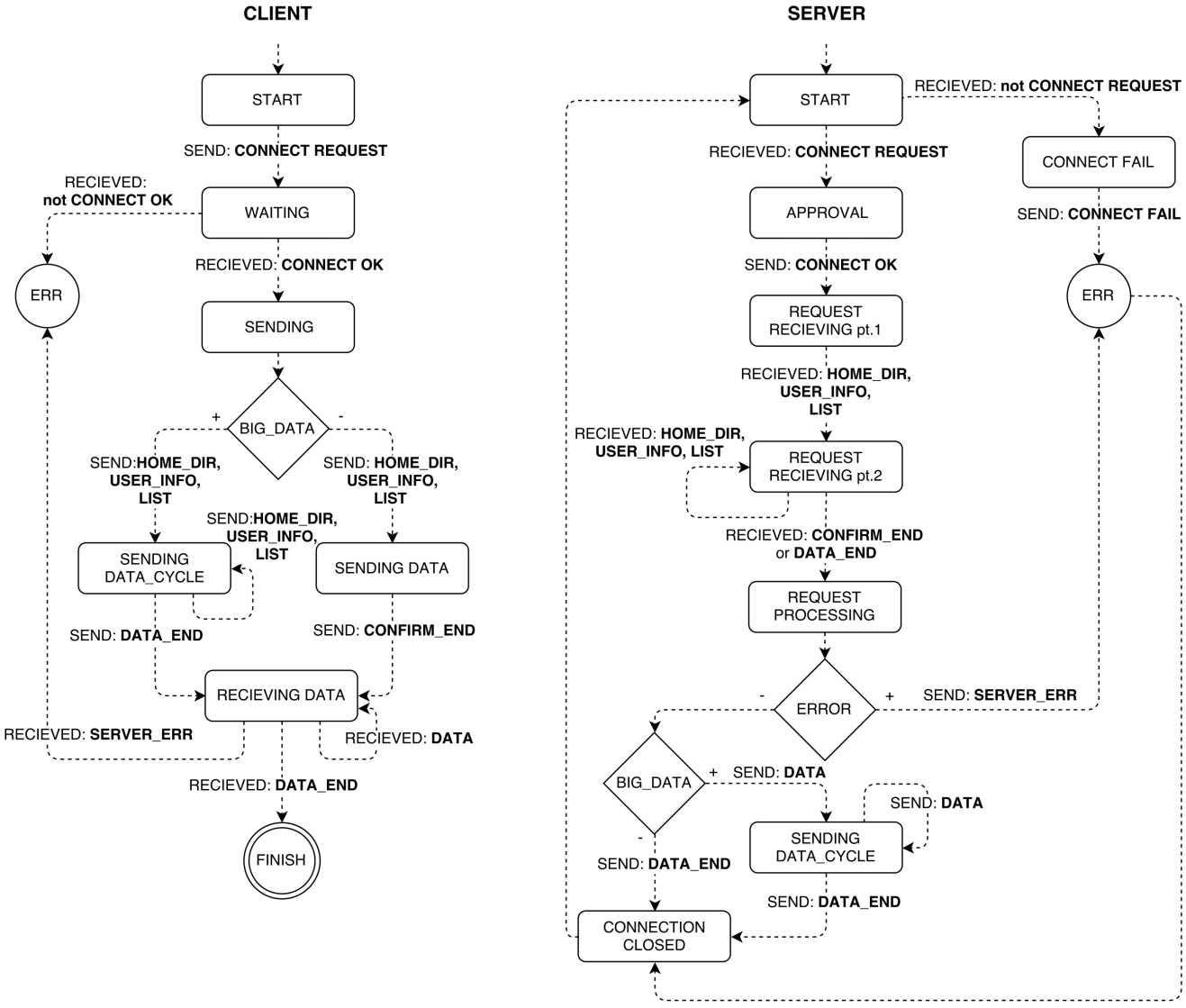
4.2 Typy správ

Protokol rozlišuje niekoľko typov správ:

- **DATA** - posielané sú užitočné dáta
- **DATA_END** - koniec posielania užitočných dát
- **SERVER_ERR** - chyba na strane servera
- **LIST** - požiadavka na zoznam užívateľov
- **HOME_DIR** - požiadavka na domovský adresár užívateľa
- **USER_INFO** - požiadavka na informácie o užívateľovi
- **CONFIRM_END** - potvrdenie ukončenia posielania, už bez užitočných dát
- **CONNECT_REQ** - žiadosť pre pripojenie
- **CONNECT_OK** - schválenie pripojenia
- **CONNECT_FAIL** - zamietnutie pripojenia

4.3 Princíp funkčnosti

Ako prvé posiela klient už bežiacemu serveru správu typu **CONNECT_REQUEST** ako požiadavku na spojenie. Ten v tomto prípade príjme správu **CONNECT_REQUEST** a odošle klientovi priaznivú odpoveď vo forme správy s typom **CONNECT_OK** ako indikátor úspešného spojenia. V inom prípade posiela naspäť klientovi neúspešnú správu **CONNECT_FAIL**. Po tejto úvodnej „dohode“ klient zahajuje posielanie požiadavky. Ak ide o príliš veľkú správu a nezmestí sa do veľkostou *BUFSIZE* definovaného buffera *data* posiela sa na viac krát a koniec posielania naznačí serveru správou **DATA_END**. V prípade menšej správy sa pošle serveru a ako naznačenie konca posielania pošle za tým správu typu **CONFIRM_END**, ktorá už neobsahuje užitočné dáta. Server príjma tieto správy a po ich prijatí sa snaží obslúžiť požiadavku klienta zberom požadovaných informácii. V prípade chyby pri hľadaní, alebo inej chyby na strane servera, posiela server klientovi správu typu **SERVER_ERR** a klient sa ukončí neúspešne. Avšak ak nedôjde k chybe, server posiela odpoveď a opäť tu záleží na jej veľkosti. **DATA_END** správu spolu s relevantnými dátami posiela v prípade kratšej správy, inak v cykle posiela správy typu **DATA** a ukončí to správou **DATA_END**. Klient rovnako v cykle obdržal informácie zo severa a úspešne sa tak uzatvára komunikácia.



Obr. 2: Konečný automat aplikačného protokolu zo strany klienta i servera.

5 Zauijímavosti implementácie

Zdrojový kód ako klientskej tak serverovej aplikácie je napísaný v jazyku C. Obsluhu viacero klientov naraz server zabezpečuje funkciou *fork()*, ktorá vytvorí nový proces duplikovaním volajúceho procesu. Detský proces tak vždy obslúži klienta a hlavný sa vracia spať čakať na prichádzajúce pripojenia. Pre zmiernenie nárokov na CPU, sa proces servera na istý čas uspí volaním fukncie *usleep()*.

Pre získanie informácie o užívateľoch zo súboru `/etc/passwd` server využíva funkcie z knižnice <*pwd.h*>. Funckie *getpwnam()* a *getpwent()* sú schopné získať potrebné dáta z `/etc/passwd` a vrátiť vo vhodnej forme aj bez explicitného volania funkcií pracujúcich so súbormi (*fopen, fread,* ...) a následneho spracovania textu.

6 Príklady použitia

Spustenie servera na `eva.fit.vutbr.cz`

```
> ./ipk-server -p 33333
```

```
$ ./ipk-client -h eva.fit.vutbr.cz -p 33333 -l xlis
xlisci01
xlisci02
xlisie00
xliska16
xlisti00
```

```
$ ./ipk-client -h eva.fit.vutbr.cz -p 33333 -n xlisci02
Liscinsky Matus,FIT BIT 2r
```

```
$ ./ipk-client -h eva.fit.vutbr.cz -p 33333 -f xlisci02
/homes/eva/xl/xlisci02
```

```
$ ./ipk-client -h eva.fit.vutbr.cz -p 33333 -f invalid
Unsuccesful request
```

Literatúra

[1] Grygarek, P.: Softwarová rozhraní systémů UNIX pro přístup k síťovým službám. online, 2016.
URL http://www.cs.vsb.cz/grygarek/LAN/sockets.html

[2] Lattrel, R.: Designing and Implementing an Application Layer Network Protocol Using UNIX Sockets and TCP. online, 2012.
URL https://www.egr.msu.edu/classes/ece480/capstone/fall12/group02/documents/Ryan-Lattrel_App-Note.pdf

[3] Wikipedia: Klient-server. online, 2017.
URL https://sk.wikipedia.org/wiki/Klient-server