VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Dokumentácia k projektu do predmetu IPK

Implementácia klient-server aplikácie

12. marca 2018

Vladimír Marcin xmarci10

Obsah

1 Úvod

Dokumentácia popisuje jednoduchú klient-server aplikáciu v C/C++ realizujúcu sprostredkovanie informácií o užívateľoch na servere. V kapitole 3 bude približený návrh a implementácia aplikačného protokolu, používaného spomínanou aplikáciou.

2 TCP

TCP (Transmission Control Protocol) je jedným z prokolov balíka internetových protokolov, ktoré tvoria jeho jadro. Vďaka TCP môžu programy na počítačoch v sieti vytvárať medzi sebou spojenia, ktorými je možné posielať dáta. Protokol pritom zaručuje, že dáta odoslané z jedného konca spojenia budú prijaté na druhej strane spojenia v rovnakom poradí a bez chýbajúcich častí. [2] Práve z tohto dôvodu sa protokol TCP javil ako najvhodnejšia možnosť pre našu aplikáciu.

3 Návrh aplikačného protokolu

Spôsob akým aplikácia prenáša dáta prostredníctvom siete je, že štruktúru packetu prevádza na pole znakov, ktoré sa následne posiela cez sieť. Keďže obe komunikujúce strany (v našom prípade klient aj server) poznajú veľkosť a štruktúru packetu je tento prevod bezpečný.

3.1 Návrh Packetu

Štruktúra packetu obsahuje dva typy dát. Prvým z nich je typ posielanej správy a druhým je pole znakov obsahujúce užitočné dáta správy. Typ správy hovorí o tom aké dáta sa nachádzajú v pripojenom poli.

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

Ukážka 1: Štruktúra packetu

3.2 Typy správ

Definovanie typu správy je jednou z najdôležitejších častí návrhu protokolu [1]. Na rozlíšenie o aký druh správy sa jedná je použitý výčtový typ. Na základe číslo tohto typu, ktoré je uložené v hlavičke packetu sa, či už klient, alebo server rozhodnú čo sa bude vykonávať.

```
enum message_type {
    C_HELLO, S_ACK, RQ_INFO, RQ_HOME, RQ_LOGINS, ERROR, S_RES, D_SEN, END
}
```

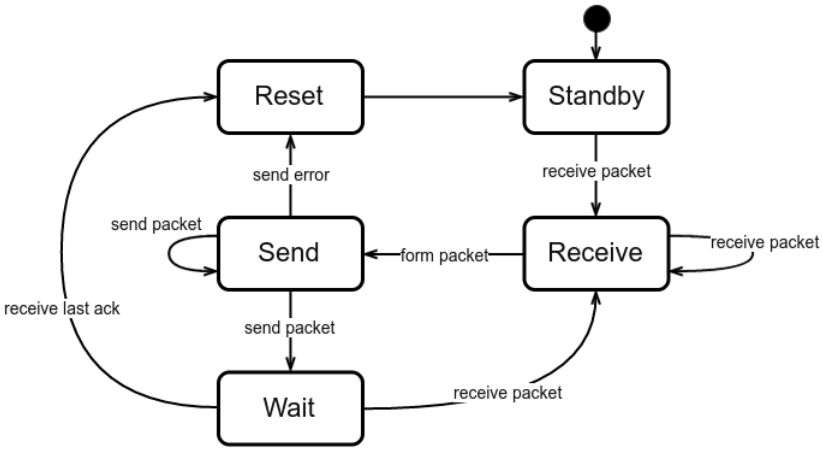
Ukážka 2: Výčtový typ message_type

- `C_HELLO` – klient hello packet slúži k identifikácii klienta
- `S_ACK` – potvrdenie od servera, že daný klient môže žiadať o obsluhu
- `RQ_INFO` – požiadavka od klienta na získanie informácií o užívateľovi (User ID Info), ktorého login je poslaný v poli `payload`
- `RQ_HOME` – požiadavka od klienta na získanie informácie o domovskom adresári užívateľa, ktorého login je poslaný v poli `payload`

- `RQ_LOGINS` – požiadavka od klienta na získanie zoznamu všetkých užívateľov, v tomto prípade je login nepovinný. Ak sa však nachádza v poli `payload`, je použitý ako prefix pre výber užívateľov
- `ERROR` – tento typ správy posiela server a indikuje, že nastala nejaká chyba pri vyhľadávaní potrebných údajov
- `S_RES` – tento typ správy posiela server a hovorí klientovi, že celá správa je doručená
- `D_SEN` – posielanie dát bude pokračovať
- `END` – tento typ správy posiela klient serveru a hovorí, že prijal všetky dáta

3.3 Server

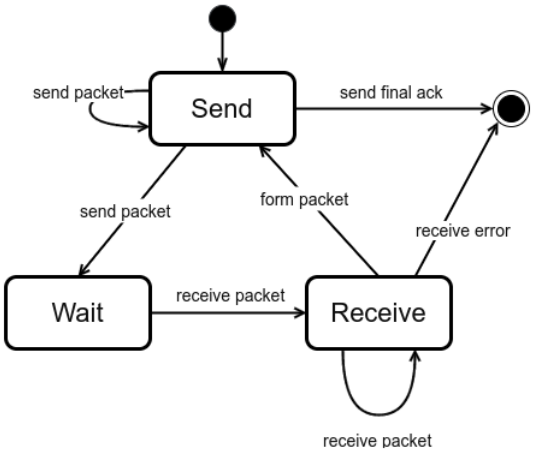
Server hneď po štarte prejde do stavu *standby* kde čaká na pripojenie klientov. V prípade, že niektorý z klientov pošle požiadavku, server prechádza do stavu *receive*. Ak je prijímaná správa väčšia ako kapacita poľa `payload` (pozná to tým, že typ správy od klienta je D_SEN) server zotrvá v stave *receive*, kým nepríjme celú správu. V momente, keď server príjme celú správu (typ správy je iný ako D_SEN) v závislosti na jej type pripravý nový packet a prechádza do stavu *send*. Znova ak je veľkosť posielaných dát väčšia ako veľkosť poľa `payload` je potrebné ich poslať po častiach. Posledná správa, ktorá bude obsahovať dáta bude označená ako `S_RES` a klient tak rozpozná, že prijal všetky dáta. Po odoslani dát server prechádza do stavu `wait`, kde čaká na ďalšiu správu od klienta alebo potvredenie od klienta, že prijal všetky dáta. Ak klient zašle toto potvrdenie (typ správy END) server prejde do stavu *reset*, v ktorom uvoľní zdroje, uzatvorí spojenie s daným klientom a prechádza do stavu `standby`, kde je znova pripravený na komunikácia s novým klientom. V prípade, že pri získavaní informácií, ktoré klient požaduje dôjde k chybe server pošle klientovi správu `ERROR` a prechádza do stavu *reset* a následne *standby*.



Obr. 1: Server – konečný automat

3.4 Klient

Klient príjme ako užívateľský vstup argumenty príkazového riadku a neočakáva žiadny užívateľský vstup počas behu. Hneď po spustení prechádza do stavu *send* a posiela požiadavku serveru. Následne prechádza do stavu *wait*, kde čaká na odpoveď od servera. Po prijatí odpovede ide do stavu *receive* kde sa na základe prijatej správy rozhodne ako bude pokračovať. Ak príjme správu typu `ERROR` ukončí svoju činnosť, inak vytvorí novú správu s potvrdením, že prijal všetky dáta a násladne sa takisto ukončí. Príjimanie a posielanie správ, ktoré sú väčšie ako veľkosť poľa `payload` je riešené rovnako ako na strane servera viz podkapitola 3.3.



Obr. 2: Klient – konečný automat

3.5 Príklad komunikácie

```
S: start
C: start

C: C_HELLO tajneheslo
S: S_ACK
C: RQ_INFO xmarci10
S: S_RES Marcin Vladimir,FIT BIT 2r
C: END

C: finish
S: close connection
```

4 Implementácia

Aplikácia bola napísaná v jazyku C/C++. Implementácia obsahuje dva rozhrania ktoré zdieľa ako klient tak aj server:

- `Socket.h` – implementácia triedy deklarovanej v tomto module sa nachádza v súbore `Socket.cpp`. Táto trieda bola s menšími úpravami prevzatá z práce [1] a slúži na prácu s Berkeley UNIX socketmi. Výhodou je, že zapuzdruje používanie funkcií nižšej vrstvy, čo v konečnom dôsledku sprehľadňuje finálny kód.
- `packet.h` – obsahuje definíciu štruktúry packetu a typu `message_type` na identifikáciu typu správy

4.1 Implementačné detaily

Na získavanie infromácií o užívateľoch bola použitá knižnica `pwd.h` a funkcia `getpwnam`.

Zaistenie obsluhy viacerých klientov súčasne je riešené pomocou procesov. Hlavný proces čaká na požiadavky od klientov. Ak nejaká príde, vytvorí detský proces, ktorý obslúži danú požiadavku a on sám čaká na požiadavky od ďalších klientov.

Keďže server beží v nekonečnom cykle pri každom opakovaní cyklu, je hlavný proces serveru pozastavený na jednu sekundu aby nespotreboval všetok procesorový čas.

5 Demonštrácia funkčnosti

```
./ipk-client -h merlin.fit.vutbr.cz -p 55546 -n xmarci10
   Marcin Vladimir,FIT BIT 2r
./ipk-client -h merlin.fit.vutbr.cz -p 55546 -f xmarci10
   /homes/eva/xm/xmarci10
./ipk-client -h merlin.fit.vutbr.cz -p 55546 -l xmar
   xmarci05
   xmarci10
   xmarek02
   xmarek59
   xmarek62
   xmarek64
   xmarek66
   xmarek67
   xmarek69
   xmares16
   xmarka01
   xmarki00
   xmarko12
   xmarko15
   xmarko17
   xmarne00
   xmarsi03
   xmarti52
   xmarti70
   xmarti76
   xmarti78
   xmarti80
   xmarti82
   xmarti83
   xmarti84
   xmarti85
   xmarti86
   xmarun00
   xmarus05
   xmarus06
   xmarus07
   xmarus08
./ipk-client -h merlin.fit.vutbr.cz -p 55546 -f invalid
   ERROR: Server error (login wasn't find)!
```

Literatúra

[1] Lattrel, R.: Designing and Implementing an Application Layer Network Protocol Using UNIX Sockets and TCP. online, 2017.
URL https://www.egr.msu.edu//classes/ece480/capstone/fall12/group02/documents/Ryan-Lattrel_App-Note.pdf

[2] Wikipedia: Protokol riadenia prenosu. online, 2018.
URL https://sk.wikipedia.org/wiki/Protokol_riadenia_prenosu