**ŽILINSKÁ UNIVERZITA V ŽILINE**

**FAKULTA RIADENIA A INFORMATIKY**

**Princípy operačných systémov (5BS101)**

Semestrálna práca

2021/22

Andrej Michalek

Vladimír Kuderavý

**Programátorská dokumentácia**

Pri navrhovaní semestrálnej práce sme chceli použiť objektový návrh, preto sme sa rozhodli ju naprogramovať v jazyku C++. Vďaka tomu bolo možné celý projekt rozdeliť do viacerých tried.

Projekt má klientsku a serverovú časť. Pri tvorbe aplikácie sme sa ju snažili navrhnúť tak, aby čo najlepšie reflektovala fungovanie reálnych chatovacích aplikácií. Celú réžiu fungovania aplikácie vrátane prihlásenia a registrácie používateľov má tak na starosti server. Klient má na starosti len komunikáciu so serverom - odosielanie správ a súborov na server a prijímanie správ zo servera. Celé fungovanie aplikácie vrátane fungovania socketov, ako aj využitie vlákien a riešené synchronizačné problémy popisujeme v nasledujúcej časti.

**Klientská časť aplikácie**

Hoci je celá semestrálna práca navrhnutá objektovo, vzhľadom na jednoduchosť klientskej časti sme sa rozhodli väčšinu logiky klienta umiestniť do súboru client.cpp.

Na začiatku v metóde main voláme funkciu socket s parametrami AF\_NET, SOCKET\_STREAM a 0, ktorá nám vráti deskriptor socketu používajúci protokol TCP/IPv4. Následne vytvoríme štruktúru sockaddr\_in a do nej nastavíme adresu servera, na ktorý sa chceme pripojiť a príslušný port. V tomto prípade to musí byť rovnaký port, aký je nastavený aj v serverovej časti aplikácie. Volaním funkcie connect nadviažeme spojenie so serverom. Funkcia vracia, či sa podarilo nadviazať spojenie a či bude možné daný socket ďalej použiť na výmenu dát so serverom, preto kontrolujeme návratovú hodnotu funkcie a ak došlo k problému, vypíše sa klientovi problem 2 a program končí.

Ak bolo spojenie úspešne nadviazané, vytvorí sa štruktúra dát (s názvom data), kde sa pošle smerník na socket. Druhým parametrom je false, pretože štruktúra ešte posiela boolean, či treba odoslať súbor, ktorý využívame v programe ďalej.

Volaním pthread\_create sa vytvorí vlákno zobrazovača, ktoré má na starosti prijímať všetky nové správy od servera a následne ich spracovať. Vlákno main následne vstupuje do while cyklu, ktorý očakáva vstup od užívateľa. Keď užívateľ niečo napíše, jeho správa sa zašifruje a odošle na server.

Prenos všetkých správ medzi serverom a klientom prebieha prostredníctvom metód send a recv. Všetky reťazce, ktoré sa odosielajú z klienta alebo servera sa pred odoslaním zašifrujú v statickej metóde zasifrujSpravu triedy Hash a všetky reťazce, ktoré sa prijímajú sa ihneď po prijatí odšifrujú pomocou funkcie odsifrujSpravu.

Vlákno zobrazovača neustále prijíma správy zo servera a následne ich po riadkoch analyzuje, či správa neobsahuje špeciálnu akciu - požiadavku na priatie alebo odoslanie súboru. Špeciálne požiadavky sa zo servera posielajú ako texty vždy na samostatnom riadku napríklad v takomto tvare: : <#nacitajSubor#>. Aby ich bolo možné jednoducho meniť, pripravili sme pre ne triedu Konštanty, kde sa všetky špeciálne akcie nachádzajú.

Ak sa prijme požiadavka na odoslanie súboru, nastaví sa premenná v zdieľanej štruktúre trebaOdoslatSubor na true. Keďže túto premennú medzi sebou zdieľajú obe vlákna klienta, ošetrili sme ju pomocou mutexu - pred prístupom k nej sa vždy mutex zamyká a po dokončení práce sa odomkne.

Vo vlákne, ktoré prijíma správy od klienta z konzoly sa vždy predtým, ako sa správa odošle na server, kontroluje, či netreba odoslať súbor na server, ak treba, volá sa metóda odoslanie súboru. Tá sa pokúsi pomocou objektu citac triedy ifstream otvoriť súbor. Ak sa to podarilo, klientovi sa zobrazí výpis, že má počkať, kým sa dokončí odoslanie súboru. Celý obsah súboru sa potom načíta, prilepí sa k nemu hlavička obsahujúca názov súboru a veľkosť súboru, ktorá sa odošle na server. Keďže mód odosielania súboru vyvoláva samotný server, užívateľ ho bez neho nemôže spustiť - do tohto bodu sa program dostane len vtedy, ak server čaká práve hlavičku súboru a presne v tomto tvare. Obsah súboru by mohol prekročiť kapacitu buffera, preto sa posiela postupne rozdelený po 4096 bajtoch a server si ho na druhej strane poskladá, pretože ho v takomto formáte očakáva. Keďže sa posiela celková veľkosť súboru, server vie, koľko častí súboru má očakávať.

Ak by užívateľ zadal zlý názov súboru, a teda by sa súbor nepodarilo otvoriť, užívateľ si môže vybrať, či chce skúsiť znova. Ak áno, môže sa opraviť, ak nie, program pokračuje ďalej bez odoslania súboru, pričom na server sa posiela o tom špeciálna správa a ten na to reaguje posunutím v rámci užívateľského prostredia do správnej sekcie.

Podobným spôsobom funguje na strane klienta aj prijímanie súborov. Ak vlákno zobrazovača zistí, že zo servera prišla požiadavka na uloženie súboru, zavolá sa metóda prijmanie súboru, ktorá postupne z prijatých dát načíta názov súboru, obsah a počet bajtov obsahu a postupne od servera prijíma celý obsah súboru. Užívateľ je informovaný, kto mu poslal súbor a v akej konverzácií mu ho poslal. Prijatý súbor sa automaticky uloží a program pokračuje ďalej.

Klientská časť programu končí vtedy, keď klient napíše do konzoly koniec - server ho na túto možnosť upozorní len v prípade, ak je na serveri odhlásený (ale program by fungoval korektne aj keby sa odpojil v inom čase). Ak napíše koniec, pomocou volania pthread\_cancel sa ukončí vlákno zobrazovača, následne sa joine (pthread\_join), aby sa uvoľnili prípadné alokované prostriedky a zavolá sa close(sock), aby sa zatvorilo pripojenie k danému socketu.

**Serverova časť aplikácie**

Serverová časť aplikácie začína v metóde main, ktorá sa nachádza v súbore server.cpp. Na začiatku sa vytvorí objekt triedy Data.

Trieda Dáta obsahuje zoznam všetkých používateľov, zoznam aktuálne prihlásených používateľov, zoznam všetkých konverzácií a zoznam otvorených socketov a mutexov (toto riešenie je vysvetlené neskôr). Zoznam všetkých používateľov máme v štruktúre unordered\_map, pretože potrebujeme často efektívne vyhľadávať používateľa podľa kľúča, ktorým je jeho meno. Ostatné zoznamy sú implementované ako vector. Keďže ide o kritickú sekciu, do ktorej môžu v niektorých prípadoch zasahovať viaceré vlákna klientov spustených na serveri súčasne, rozhodli sme sa dať do atribútov aj mutex. Klientské vlákno do triedy dáta pristupuje len pomocou volaní jej funkcií a v samotných funkciách si trieda Dáta sama zamykaním a odomykaním mutexov zabezpečuje, aby nemohli dvaja klienti súčasne prepisovať objekty v kritickej sekcii. Týmto spôsobom sme vyriešili synchronizačný problém - pristupovanie k zdieľanej štruktúre od viacerých vlákien klientov na serveri.

Po vytvorení objektu dát sa nad ním zavolá metóda nacitajVsetko. Tá načíta zo súboru všetky dáta o užívateľoch - ich mená a heslá a zároveň aj informácie o priateľstvách. Všetky heslá sa ihneď po priradení príslušnému užívateľovi šifrujú a hash sa vyberá náhodne, preto ak bude mať viacero používateľov rovnaké heslo, v súbore bude uložené ako iné heslo. Trieda používateľ dokáže pravosť hesla overiť pomocou funkcie na overenie dvoch hesiel z Hashovacej triedy, pričom heslo vždy overí voči všetkým možným hashom.

V metóde main sa po načítaní dát vytvorí vlákno vytvárač klientov, ktorému sa odovzdá referencia na štruktúru dát a referencia na vytvorený vector spustených vlákien klientov. Ten potrebujeme, aby bolo možné aplikáciu na strane servera korektne ukončiť. Trieda main totiž na serveri čaká na napísanie slova “koniec”. Ak ho niekto zadá, zamkne mutex na dátach (aby zabezpečila, že sú v tej dobe dáta v konzistentnom stave), ukončí všetky vlákna klientov aj vlákno vytvárača klientov, následne odomkne mutex (vtedy už žije len toto vlákno - main) a zavolá metódu zatvorVsetkyOtvoreneSockety na dátach, aby sa ešte pred skončením aplikácie korektne zatvorili všetky otvorené sockety. Na koniec sa zavolá metóda ulož všetko a aplikácia končí.

Funkcia vytvárač klientov, ktorá sa volá po vytvorení vlákna vytvárača klientov, na začiatku vytvorí sieťový socket volaním funkcie socket. Skontroluje sa, či sa sokcet podarilo vytvoriť a ak áno, zavolá sa funkcia bind, ktorá socketu pridá adresu a port. Opäť skontrolujeme, či pridelenie prebehlo bez problémov a ak áno, zavoláme funkciu listen, aby mohol socket začať prijímať spojenia od klientov.

Následne vlákno vstupuje do while cyklu, vďaka ktorému je možné, aby sa na server pripojilo viacero klientov. Na začiatku while cyklu je volaná funkcia accept, ktorá čaká na prijatie požiadavky o vytvorenie spojenia zo strany klienta. Ak sa pripojí klient, funkcia vráti deskriptor socketu pomocou ktorého môžeme s klientom komunikovať a program pokračuje ďalej. Na strane servera sa vypíše, že bol pripojený klient a následne sa mu vytvorí a inicializuje vlastný mutex. Potom sa vytvorí klientovi nové vlákno, do ktorého sa okrem jeho deskriptora socketu a mutexu pošle aj referencia na zdieľanú štruktúru dáta. Vytvorené vlákno sa pridá do zoznamu otvorených vlákien a vlákno vytvárač klientov čaká vďaka while cyklu opäť na metóde accept, kým sa pripojí ďalší klient.

Na začiatku klientského vlákna na serveri (funkcia vlaknoFunkcia) sa vytvorí objekt typu SocketAMutex a pridá sa doňho socket a mutex daného klienta. Ten sa následne pridá pomocou metódy pridajOtvorenySocket do zoznamu v objekte dáta. Následne sa inicializuje booleanovská premenná odpojilo ho na false, aby bolo možné korektne ukončiť while cyklus. While cyklus sme v tejto časti programu použili preto, aby sa na používateľovom počítači po odhlásení mohol prihlásiť alebo registrovať ďalší používateľ (napríklad jeho rodinný príslušník). V opačnom prípade by server musel užívateľa po odhlásení hneď odpojiť, čo si myslíme, že by bolo horšie riešenie, keďže si klient môže sám vybrať, ak chce aplikáciu ukončiť.

Prihlásenie klienta má na starosti trieda Prihlásenie. Klientské vlákno ihneď volá jej metódu prihlasenie, kde sa môže užívateľ ihneď po pripojení na server prihlásiť do existujúceho konta alebo zaregistrovať a vytvoriť tak nové konto.

Ak s klientom komunikuje jeho užívateľské prostredie (tj. odosiela mu cez socket správu priamo jeho vlákno bežiace na serveri), tak na komunikáciu používame triedu Pomocna. Tá má metódu prijmi správu, ktorá cez recv dokáže prijať správu do poľa, pričom ho pred prijatím automaticky nuluje a dokáže aj zachytiť chyby v spojení - ak sa klient odpojil alebo spojenie vypadlo, tak vráti 1. Ak došlo k úspešnému prijatiu správy, tak vráti 0. Túto metódu voláme vždy cez if s returnom, takže ak sa klient kedykoľvek odpojí, server na to hneď zareaguje a automaticky ho odhlási a korektne ukončí jeho vlákno na serveri.

Metóda odosliSpravu v triede Pomocna posiela správu cez send, ešte predtým ju zašifruje. Pred volaním send sa zamkne komunikačný mutex daného používateľa a po odoslaní sa odomkne. Každý používateľ má vlastný mutex na svojom komunikačnom sockete kvôli tomu, že keď sa odosiela veľmi veľký súbor, posiela sa na niekoľkokrát. Na prijatie celého súboru sa čaká v cykle, preto sme museli zabezpečiť, aby niekto do daného socketu neposielal v tom čase iné správy. Ak by to nebolo ošetrené, mohlo by sa stať, že medzitým, ako klientska časť čaká na prijatie súboru, iné vlákno by danému klientovi odoslalo správu a prijímanie súboru by bolo narušené. Ak by bol jeden spoločný komunikačný mutex pre všetkých klientov, výrazne by to spomalilo komunikáciu - v jednu dobu by mohol server posielať správu len jednému klientovi a mať otvorených viacero vlákien by stratilo význam. Preto sme sa rozhodli, že každý klient bude mať vlastný komunikačný mutex, ktorý sa bude zamykať, ak bude niekto chcieť odoslať správu danému používateľovi. Mutex je tak viazaný len na daný klientov deskriptor socketu, takže keď sa stratí spojenie, zavolá sa metóda pthread\_mutex\_destroy a následne sa vymaže.

Trieda prihlásenie tak klientovi pošle správu, že má možnosť sa prihlásiť, zaregistrovať alebo ukončiť aplikáciu. Ak klient zadá 1 - prihlásenie, volá sa metóda prihlasSa, kde si server od neho vypýta používateľské meno a následne heslo. Zavolá sa metóda prihlas v triede Data, kde sa zamkne hlavný mutex, skontroluje sa, či už nie je prihlásený a ak áno, odomkne sa mutex a do predchádzajúcej metódy sa vráti text „už ste prihlásený z iného počítača“, ktorý sa používateľovi zobrazí ako dôvod, prečo odoslanie zlyhalo. Ak ešte nie je prihlásený, skontroluje sa heslo a ak je dobré, používateľ sa pridá do poľa prihlásených, odomkne sa mutex a metóda končí. Ak heslo nesedí, táto metóda vráti, že používateľ zadal zlé meno alebo heslo. Metódy v triede Data vrátane tejto sme implementovali tak, aby oni sami nekomunikovali s používateľom, ale len vracali nejakú informáciu do používateľského rozhrania, ktoré s ním komunikuje. Vďaka tomu môže byť hlavný mutex uzavretý len veľmi krátky čas a nespomaľuje sa chod aplikácie pre ostatných klientov.

V triede prihlásenie sa nachádza aj metóda registrujsa, ktorá od užívateľa vyžiada jeho meno a heslo. Na prístup ku kritickej sekcii opäť použije metódu triedy Data, tentokrát registruj, ktorá funguje obdobným spôsobom ako prihlas. Po registrácií používateľa sa tento používateľ môže prihlásiť do svojho účtu.

Metóda prihlásenia v triede Prihlasenie tak musí mať objekt Používateľa, ktorý získala od triedy Data jeho úspešným prihlásením, tento objekt vráti vláknu klienta, ktoré ju volalo. Pri ďalšej práci s týmto objektom sme si dali pozor, aby sme nikdy nepristupovali k jeho dátam priamo, ak sa objekt použije, tak len ako parameter do triedy Data, ktorá pred prístupom doňho zamkne hlavný mutex, a potom ho odomkne, aby sa zachovala ucelenosť kritickej sekcie.

Po prihlásení používateľa ďalej voláme metódu vyberSiMoznost v triede Možnosti - dá sa povedať, že táto trieda je pre klienta vstupným bodom do samotnej chatovacej aplikácie a zastrešuje užívateľské rozhranie.

Klientovi sa zobrazia možnosti, kde si môže vybrať, čo chce robiť. Prácu s priateľmi z pohľadu užívateľského rozhrania spravuje trieda SpravaPriatelstiev a prácu so správami a súbormi spravuje SpravaKonverzacii.

Správa priateľstiev má tri metódy - posliZiadostOPriatelstvo, odoberZPriatelov a potvrdenieAleboZamietnutieZiadosti.

PosliZiadostOPriatelstvo spolupracuje s metódou posliZiadostOPriatelstvo v triede Data, kde sa nájde používateľ, ktorému sa má žiadosť poslať, žiadosť sa pridá do jeho žiadostí a následne sa zavolá správa odosliSpravuCezSocket, aby sa užívateľovi poslala správa, že má novú žiadosť o priateľstvo. Ak chce trieda Data posielať správu používateľom, vždy ju posiela cez metódu odosliSpravuCezSocket. Tá zamkne hlavný mutex a skontroluje, či je používateľ prihlásený. Ak nie je, správa sa pridá do jeho neprečítaných správ a zobrazí sa mu ihneď po prihlásení. Ak je prihlásený, získa sa mutex jeho socketu a ten sa zamkne, potom sa odomkne hlavný mutex, odošle sa mu správa a odomkne sa mutex jeho socketu.

Týmto spôsobom zamykania mutexov daných socketov sú ošetrené všetky situácie - nemôže sa stať, že viacero používateľov chce jednému používateľovi poslať napríklad súbor alebo správu a on sa medzitým odpojí, jeho mutex zanikne a dôjde k problémom s aplikáciou. Ak by k takejto situácií došlo, hlavný mutex zaručí, že v rade na odomknutie používateľovho mutexu nemôže čakať viac vlákien. Naopak, vlákno používateľa nemusí na hlavný mutex čakať a tak môžu používatelia pracovať paralelne so svojím užívateľským rozhraním bez problémov. Takisto ostatní užívatelia nemusia čakať, kým server odošle celé správy, prípadne celé súbory, lebo v rámci odosielania nie je zamknutý hlavný mutex, ale len mutex socketu konkrétneho používateľa. Server tak dokáže obsluhovať viacero klientov paralelne.

Podobným spôsobom fungujú aj metóda odoberZPriatelov - klientovi sa najskôr zobrazí zoznam jeho priateľov, z ktorého môže odobrať niektorého z nich. Ten je získaný z metódy dajZoznamPriatelov z triedy Data a následne si môže vybrať jedného z nich. Informácia sa spracuje v metóde spracujOdobratieZPriatelov v triede Data, pričom sa dbá, aby tieto metódy pristupovali do kritickej sekcie len so zamknutým mutexom. Potvrdenie alebo zamietnutie žiadosti funguje rovnakým spôsobom - užívateľ si vyberie, na ktorú žiadosť chce reagovať a následne, či ju chce potvrdiť alebo zamietnuť a jej spracovanie sa prenechá metóde spracujZiadostOPriatelstvo v triede Data.

V správe konverzácií môže používateľ vybrať existujúcu konverzáciu a v nej odoslať správu alebo vytvoriť novú konverzáciu a popridávať do nej svojich priateľov. Správa sa vždy odošle všetkým účastníkom konverzácie, ktorých má užívateľ, ktorý správu posiela, vo svojich priateľoch.

Na odoslanie textových správ sa posiela metóda posliSpravu v triede Data, ktorá prechádza všetkých členov konverzácie a zisťuje, či má daného člena odosielateľ v priateľoch. Ak má zavolá sa metóda posliSpravuCezSocket.

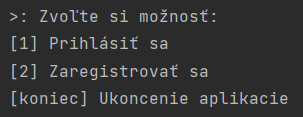
Metóda pošli súbor v Správe konverzácií odošle na klienta konštantu, ktorá spustí načítavanie súboru na klientovej strane a následne čaká, čo príde ako odpoveď od klienta. Ak príde ako odpoveď konštanta - súbor sa nepodarilo načítať, vráti klienta späť do výberu konverzácie. V opačnom prípade to znamená, že klient začal odosielať súbor a táto metóda ho začne postupne prijímať - zaznamená hlavičku súboru - jeho názov a počet znakov a na základe nej začne prijímať jednotlivé časti, ak sa celý súbor nezmestí do buffera. Po tom, ako došlo k načítaniu celého súboru, zavolá z triedy Data metódu pošli súbor, ktorá funguje podobne ako pošli správu - odošle súbor každému členovi konverzácie, ktorého má odosielateľ vo svojich priateľoch s využitím metódy odošliSuborCezSocket. Tá zabezpečí, aby bol mutex nad adresátovým socketom zamknutý - teda aby sa poslal celý súbor bez prerušenia, ale hlavný mutex môže byť odomknutý, čiže odosielanie súboru nijako nespomalí chod servera.

V možnostiach je okrem správy priateľstiev, správy konverzácií a odhlásenia aj možnosť zmazať účet. Najskôr sa zavolá metóda odstráň používateľovi všetkých priateľov z triedy Data, ktorá vymaže používateľovi všetkých priateľov a všetci jeho priatelia sú informovaní o tom, že bol odstránený z dôvodu zrušenia účtu, následne sa používateľ odhlási a zavolá sa odstráň účet (obe metódy sú z triedy Data, takže je v nich ošetrený prístup do kritickej sekcie).

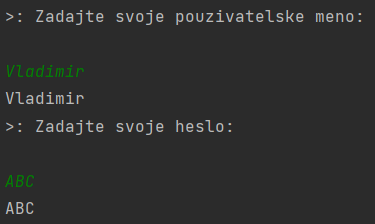
Ak klient ukončí spojenie, z možností sa vystúpi s návratovou hodnotou true, ktorá sa nastaví do premennej odpojilo ho. To ukončí while cyklus klientovho vlákna na serveri, vymaže otvorený socket z dát a vlákno končí.

**Používateľská príručka**

Na začiatku spustenia aplikácie klienta sa používateľovi zobrazí menu, kde si môže vybrať či sa chce prihlásiť, registrovať alebo chce ukončiť aplikáciu.

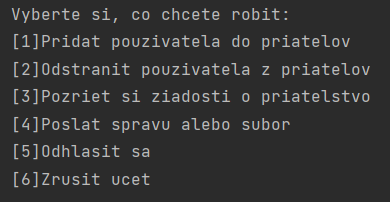


V prípade, že používateľ ešte nemá vytvorený účet, musí sa najprv zaregistrovať. Po odoslaní voľby číslo 2 sa používateľovi zobrazí správa zo servera, ktorá ho požiada aby zadal používateľské meno. V prípade, že zadané meno už nepoužíva niekto iný, sa používateľovi zobrazí požiadavka zo servera, aby zadal heslo. V opačnom prípade sa server spýta používateľa, či chce opakovať zadanie používateľského mena. Po zadaní hesla, sa používateľovi zobrazí správa, že registrácia prebehla úspešne v prípade, že nedošlo k chybe v spojení klienta a servera.



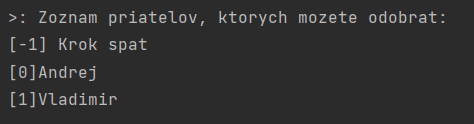
Po úspešnom zaregistrovaní sa môže používateľ prihlásiť tak, že zadá voľbu číslo 1. Server vyzve používateľa aby zadal svoje prihlasovacie meno a heslo. V prípade, že zadal zlé meno alebo heslo, server ho o tom informuje a môže prihlásenie opakovať.

Keď sa používateľ prihlási zobrazí sa mu menu, kde si môže pridať iného používateľa do priateľov, odstrániť používateľa z priateľov, pozrieť si svoje aktuálne žiadosti o priateľstvo, poslať správu iným používateľom, odhlásiť sa alebo zrušiť účet.



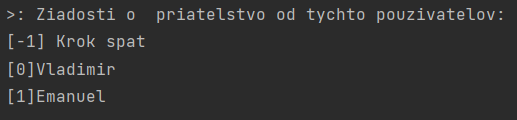
V prípade vybratia prvej možnosti si server vypýta meno používateľa ktorému chce poslať žiadosť o priateľstvo. Po zadaní správneho mena sa danému používateľovi pošle správa o priateľstvo a vypíše sa správa o úspešnom poslaní žiadosti.

Pokiaľ si používateľ vyberie druhú možnosť server ponúkne používateľovi zoznam priateľov ktorých si môže odstrániť.

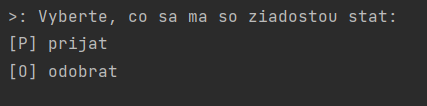


Po správnom zadaní voľby sa zo zoznamu priateľov odoberie vybraný používateľ.

Pokiaľ používateľ zadá možnosť číslo 3 v hlavnom menu, má možnosť si pozrieť aktuálne žiadosti o priateľstvo, ktoré mu poslali ostatní používatelia.

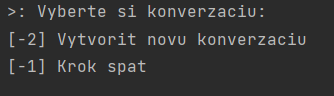


Po vybratí danej žiadosti o priateľstvo má používateľ možnosť danú žiadosť prijať alebo žiadosť odmietnuť a odobrať.

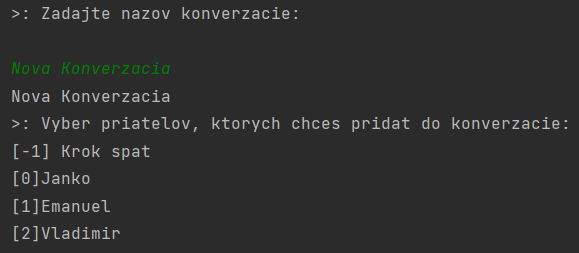


V prípade, že chce používateľ poslať správu alebo súbor inému používateľovi alebo poslať správu alebo súbor viacerým používateľom naraz v spoločnej konverzácií, musí si v hlavnom menu zvoliť možnosť číslo 4.

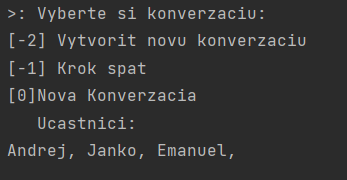
Používateľovi sa zobrazí zoznam všetkých konverzácií. Pokiaľ používateľ nemá vytvorenú žiadnu konverzáciu, musí si pred poslaním správy najprv nejakú konverzáciu vytvoriť a vybrať používateľov, ktorý majú byť účastníci konverzácie.



Pri vytvorení novej konverzácie si server vypýta od používateľa názov konverzácie a následne vypíše priateľov ktorých môže používateľ pridať do konverzácie.



Následne si používateľ môže vybrať zo zoznamu konverzácií danú konverzáciu.

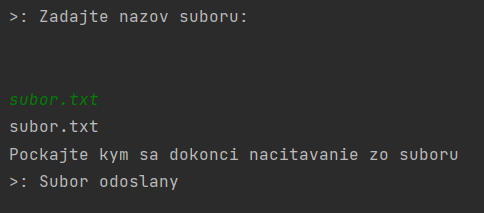


Po vybraní konverzácie sa používateľovi zobrazí možnosť poslať správu alebo poslať súbor.

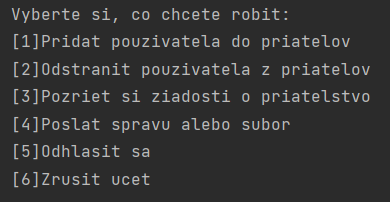


Pokiaľ si používateľ vyberie možnosť poslania správy, následne stačí už len napísať do konzoly obsah správy a táto správa sa pošle všetkým účastníkom, ktorí sú pridaní v konverzácií.

V prípade, že si používateľ vyberie možnosť poslania súboru, server si od používateľa vypýta názov súboru ktorý chce poslať. Následne prebehne načítavanie súboru a používateľovi sa oznámi keď sa súbor úspešne načíta a odošle.



V hlavnom menu má používateľ ako piatu možnosť odhlásenia, kedy sa používateľ odhlási z aktuálneho účtu a môže sa prihlásiť na iný účet, vytvoriť nový, alebo ukončiť aplikáciu.



Poslednou možnosťou v hlavnou menu, je možnosť zrušiť účet, kedy sa odstráni účet používateľa zo servera. V prípade, že si používateľ zvolí túto možnosť, server sa ho ešte spýta, či chce naozaj zrušiť účet, aby nedošlo k nechcenému zrušeniu účtu.

Poznámka:

Pred odoslaním súborov je nutné v triede konštanty nastaviť cestu, odkiaľ sa budú súbory odosielať (metóda getRelativnaCesta()).

Link na git repozitár:

<https://github.com/VladimirKuderavy/SemestralnaPracaPOS>