Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Операционные системы»**

Студент: Смирнов А.В.

Группа: М8О-207Б-21

Вариант: 22

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/Liguha/OS>

**Постановка задачи**

**Цель работы**

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

**Задание**

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Клиент-серверная система для передачи мгновенных сообщений. Базовый функционал должен быть следующим: клиент может присоединиться к серверу, введя логин; клиент может отправить сообщение другому клиенту по его логину; клиент в реальном времени принимает сообщения от других клиентов; необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи pipe'ов.

**Общие сведения о программе**

Серверная часть компилируется из файла server.cpp, клиентская из client.cpp. Обе программы используют заголовочные файлы check\_err.hpp, constants.hpp, instructions.hpp. Используются следующие системные вызовы:

1. mkfifo() – создание именованного канала
2. unlink() – удаление имени из файловой системы
3. open() – открытие файла
4. close() – закрытие файла
5. write() – запись последовательности байт
6. read() – чтение последовательности байт

Помимо этого, используются библиотечные вызовы для работы с потоками, семафорами и мьютексами.

**Общий метод и алгоритм решения**

Программа использует 3 «типа» пайпов, первый пайп единственный в своём роде – используется для отправки незарегистрированным запроса на регистрацию, также этот пайп используется для отправки запроса на отключение от сервера, создаётся сервером при запуске и имеет постоянное имя; второй тип - пайпы для отправки сообщений серверу от зарегистрированного пользователя (запрос на передачу сообщения, создания чата и т.д.), создаются клиентом и именуются как id клиента (его pid) + заданный постфикс; третий тип - пайпы для получения ответов от сервера, создаются по аналогии со вторым типом.

Сервер узнаёт о существовании пользователя после первого запроса login (не важно удачного или же нет). Все известные пользователи хранятся в map с ключом – int (id пользователя), значением – user\*, где user – структура, хранящая информацию о пользователе и потоке, который читает пайп второго типа данного пользователя. В случае с личными сообщениями поток-обработчик отправителя «перекладывает» сообщение из пайпа сообщений сервера отправителя в пайп ответов получателя.

Чаты хранятся в виде структуры, содержащей название чата и имена пользователей этого чата, система обработки сообщений в чатах схожа с системой личных сообщений с отличием в том, ответ перекладывается в пайпы 3 типа всех пользователей чата. Добавлять в чат может любой пользователь, состоящий в чате.

Итак список команд для работы с системой получился следующим:

* login USR – авторизация под именем USR
* send private/chat DST – отправка личного/группового сообщения пользователю/чату DST
* chat create CHT – создание чата с названием CHT
* chat add CHT USR – добавление в чат CHT пользователя USR
* logout – выйти из текущего аккаунта, освободив его
* exit – отключение от сервера, завершение работы

**Исходный код**

|  |
| --- |
| **check\_err.hpp** |
| #ifndef CHECK\_ERR\_HPP  #define CHECK\_ERR\_HPP  #include <iostream>  #define CHECK\_ERROR(expr, stream, act) \  do \  { \  int res = (expr); \  if (res == -1) \  { \  std::cerr << stream; \  act; \  } \  } while (0)  #define CHECK\_ERROR\_PTHREAD(expr, stream) \  do \  { \  int res = (expr); \  if (res != 0) \  { \  std::cerr << stream; \  return -1; \  } \  } while (0)  #endif |

|  |
| --- |
| **constants.hpp** |
| #ifndef CONSTANTS\_HPP  #define CONSTANTS\_HPP  #include <string>  using namespace std;  enum query\_id  {  LOGIN,  LOGIN\_OK,  LOGIN\_ERR,  SEND\_PRIVATE,  SEND\_ERR,  GET\_PRIVATE,  SEND\_GROUP,  GET\_GROUP,  CREATE\_GROUP,  CREATE\_G\_OK,  CREATE\_G\_ERR,  ADD\_TO\_GROUP,  ADD\_G\_OK,  ADD\_G\_ERR,  LOGOUT,  EXIT  };  const string sem\_name = "data\_semaphore";  const string data\_pipe = "login\_data";  const string send\_postfix = "\_send";  const string get\_postfix = "\_get";  #endif |

|  |
| --- |
| **instructions.hpp** |
| #ifndef INSTRUCTIONS\_HPP  #define INSTRUCTIONS\_HPP  #include "unistd.h"  #include "sys/stat.h"  #include "fcntl.h"  #include "constants.hpp"  #include "check\_err.hpp"  #include <string>  using namespace std;  struct user\_info  {  string username;  int user\_id;  };  struct message  {  string author = "";  string content = "";  string channel = "";  };  int send\_message(int pipe, message msg, query\_id id)  {  int ok = 0;  CHECK\_ERROR(write(pipe, &id, sizeof(query\_id)), "", ok = -1);  int len = msg.author.length();  CHECK\_ERROR(write(pipe, &len, sizeof(int)), "", ok = -1);  CHECK\_ERROR(write(pipe, msg.author.c\_str(), len), "", ok = -1);  len = msg.content.length();  CHECK\_ERROR(write(pipe, &len, sizeof(int)), "", ok = -1);  CHECK\_ERROR(write(pipe, msg.content.c\_str(), len), "", ok = -1);  len = msg.channel.length();  CHECK\_ERROR(write(pipe, &len, sizeof(int)), "", ok = -1);  CHECK\_ERROR(write(pipe, msg.channel.c\_str(), len), "", ok = -1);  return ok;  }  message receive\_msg(int pipe)  {  message res;  int n;  read(pipe, &n, sizeof(int));  char\* str = (char\*)calloc(n, sizeof(char));  read(pipe, str, n);  res.author = str;  free(str);  read(pipe, &n, sizeof(int));  str = (char\*)calloc(n, sizeof(char));  read(pipe, str, n);  res.content = str;  free(str);  read(pipe, &n, sizeof(int));  str = (char\*)calloc(n, sizeof(char));  read(pipe, str, n);  res.channel = str;  free(str);  return res;  }  struct group\_modify  {  string group = "";  string username = "";  };  int send\_group\_modify(int pipe, group\_modify mod, query\_id id)  {  int ok = 0;  CHECK\_ERROR(write(pipe, &id, sizeof(query\_id)), "", ok = -1);  int n = mod.group.length();  CHECK\_ERROR(write(pipe, &n, sizeof(int)), "", ok = -1);  CHECK\_ERROR(write(pipe, mod.group.c\_str(), n), "", ok = -1);  n = mod.username.length();  CHECK\_ERROR(write(pipe, &n, sizeof(int)), "", ok = -1);  CHECK\_ERROR(write(pipe, mod.username.c\_str(), n), "", ok = -1);  return ok;  }  group\_modify receive\_group\_modify(int pipe)  {  group\_modify res;  int n;  read(pipe, &n, sizeof(int));  char\* str = (char\*)calloc(n, sizeof(char));  read(pipe, str, n);  res.group = str;  free(str);  read(pipe, &n, sizeof(int));  str = (char\*)calloc(n, sizeof(char));  read(pipe, str, n);  res.username = str;  free(str);  return res;  }  #endif |

|  |
| --- |
| **client.cpp** |
| #include "unistd.h"  #include "sys/stat.h"  #include "fcntl.h"  #include "semaphore.h"  #include "pthread.h"  #include "constants.hpp"  #include "instructions.hpp"  #include "check\_err.hpp"  #include <iostream>  #include <string>  using namespace std;  bool ACTIVE\_THD = true;  bool ACTIVE\_MAIN = true;  void\* get\_thd(void\* ptr)  {  user\_info\* info = (user\_info\*)ptr;  string str\_id = to\_string(info->user\_id);  int pipe = open((str\_id + get\_postfix).c\_str(), O\_RDWR);  CHECK\_ERROR(pipe, "Error: can't open pipe" << endl, ACTIVE\_THD = false);  while (ACTIVE\_THD)  {  query\_id id;  CHECK\_ERROR(read(pipe, &id, sizeof(id)), "Error: pipe read error" << endl, ACTIVE\_THD = false);  switch (id)  {  case LOGIN\_OK:  {  cout << "Logged-in" << endl;  break;  }  case LOGIN\_ERR:  {  info->username = "";  cerr << "Error: user already logged-in" << endl;  break;  }  case SEND\_ERR:  {  cerr << "Error: can't find receiver" << endl;  break;  }  case GET\_PRIVATE:  {  message msg = receive\_msg(pipe);  cout << "Private message from " << msg.author << ": " << msg.content << endl;  break;  }  case GET\_GROUP:  {  message msg = receive\_msg(pipe);  cout << "Message in chat " << msg.channel << " from " << msg.author << ": " << msg.cotent << endl;  break;  }  case CREATE\_G\_OK:  {  cout << "Success create chat" << endl;  break;  }  case CREATE\_G\_ERR:  {  cerr << "Error: can't create such chat" << endl;  break;  }  case ADD\_G\_OK:  {  int n;  CHECK\_ERROR(read(pipe, &n, sizeof(int)), "Error: pipe read error" << endl, ACTIVE\_THD = false);  char\* str = (char\*)calloc(n, sizeof(char));  CHECK\_ERROR(read(pipe, str, n), "Error: pipe read error" << endl, ACTIVE\_THD = false);  cout << "Welcome in chat " << str << endl;  free(str);  break;  }  case ADD\_G\_ERR:  {  cerr << "Error: can't add this user in this chat" << endl;  break;  }  case EXIT:  {  close(pipe);  unlink((to\_string(info->user\_id) + get\_postfix).c\_str());  unlink((to\_string(info->user\_id) + send\_postfix).c\_str());  ACTIVE\_MAIN = false;  return NULL;  }  }  }  close(pipe);  unlink((to\_string(info->user\_id) + get\_postfix).c\_str());  unlink((to\_string(info->user\_id) + send\_postfix).c\_str());  ACTIVE\_MAIN = false;  return NULL;  }  int main()  {  user\_info info;  info.username = "";  info.user\_id = getpid();  pthread\_t thd;  int data = open(data\_pipe.c\_str(), O\_RDWR);  sem\_t\* sem = sem\_open(sem\_name.c\_str(), O\_RDWR);  string str\_id = to\_string(info.user\_id);  string str\_send, str\_get;  str\_send = str\_id + send\_postfix;  str\_get = str\_id + get\_postfix;  unlink(str\_send.c\_str());  unlink(str\_get.c\_str());  CHECK\_ERROR(mkfifo(str\_send.c\_str(), S\_IREAD | S\_IWRITE), "Error: mkfifo error\n", return -1);  CHECK\_ERROR(mkfifo(str\_get.c\_str(), S\_IREAD | S\_IWRITE), "Error: mkfifo error\n", return -1);  CHECK\_ERROR\_PTHREAD(pthread\_create(&thd, NULL, get\_thd, &info), "Error: error of creating thread\n");  CHECK\_ERROR\_PTHREAD(pthread\_detach(thd), "Error: error of detach thread\n");  int pipe = open(str\_send.c\_str(), O\_RDWR);  CHECK\_ERROR(pipe, "Error: can't open pipe\n", return -1);  while (ACTIVE\_MAIN)  {  string command;  query\_id id;  cin >> command;  if (!ACTIVE\_MAIN)  return -1;  if (command == "login")  {  id = LOGIN;  string name;  cin >> name;  if (info.username != "")  {  cerr << "Error: you are already logged-in" << endl;  continue;;  }  if (name == "")  {  cout << "Please, enter correct name" << endl;  continue;  }  info.username = name;  sem\_wait(sem);  CHECK\_ERROR(write(data, &id, sizeof(query\_id)), "Error: data pipe writing\n", return -1);  CHECK\_ERROR(write(data, &info.user\_id, sizeof(int)), "Error: data pipe writing\n", return -1);  int len = info.username.length();  CHECK\_ERROR(write(data, &len, sizeof(int)), "Error: data pipe writing\n", return -1);  CHECK\_ERROR(write(data, info.username.c\_str(), len), "Error: data pipe writing\n", return -1);  sem\_post(sem);  }  if (command == "send")  {  if (info.username == "")  {  cout << "Please, login in the system" << endl;  continue;  }  string type, dst, str;  cin >> type >> dst;  if (type != "private" && type != "chat")  {  cerr << "Error: unknown message type " << type << endl;  continue;  }  getline(cin, str);  message msg;  msg.author = info.username;  msg.content = str;  msg.channel = dst;  if (type == "private")  id = SEND\_PRIVATE;  if (type == "chat")  id = SEND\_GROUP;  CHECK\_ERROR(send\_message(pipe, msg, id), "Error: sending message\n", return -1);  }  if (command == "chat")  {  if (info.username == "")  {  cout << "Please, login in the system" << endl;  continue;  }  string act, group;  cin >> act >> group;  if (act != "create" && act != "add")  {  cerr << "Error: unknown chat action" << endl;  continue;  }  group\_modify mod;  mod.group = group;  if (act == "create")  {  mod.username = info.username;  id = CREATE\_GROUP;  }  if (act == "add")  {  cin >> mod.username;  id = ADD\_TO\_GROUP;  }  CHECK\_ERROR(send\_group\_modify(pipe, mod, id), "Error: modify chat\n", return -1);  }  if (command == "logout")  {  if (info.username == "")  {  cout << "Please, login in the system" << endl;  continue;  }  id = LOGOUT;  CHECK\_ERROR(write(pipe, &id, sizeof(query\_id)), "Error: pipe writing\n", return -1);  info.username = "";  cout << "Logged-out" << endl;  }  if (command == "exit")  {  if (info.username != "")  {  id = LOGOUT;  write(pipe, &id, sizeof(query\_id));  }  id = EXIT;  sem\_wait(sem);  write(data, &id, sizeof(query\_id));  write(data, &info.user\_id, sizeof(int));  sem\_post(sem);  close(pipe);  close(data);  sem\_close(sem);  while (ACTIVE\_MAIN)  continue;  return 0;  }  }  return -1;  } |

|  |
| --- |
| **server.cpp** |
| #include "unistd.h"  #include "sys/stat.h"  #include "fcntl.h"  #include "semaphore.h"  #include "pthread.h"  #include "constants.hpp"  #include "instructions.hpp"  #include "check\_err.hpp"  #include <iostream>  #include <string>  #include <set>  #include <map>  using namespace std;  void\* user\_thd(void\*);  struct user  {  user\_info info;  pthread\_mutex\_t mutex;  int pipe\_from, pipe\_to;  pthread\_t thd;  user(string login = "", int id = 0)  {  info.username = login;  info.user\_id = id;  pipe\_to = open((to\_string(id) + get\_postfix).c\_str(), O\_RDWR);  pipe\_from = open((to\_string(id) + send\_postfix).c\_str(), O\_RDWR);  CHECK\_ERROR(min(pipe\_to, pipe\_from), "Error: pipe of " << id << '\n', return);  CHECK\_ERROR(pthread\_mutex\_init(&mutex, NULL), "Error: Gmutex error\n", return);  if (pthread\_create(&thd, NULL, user\_thd, this) != 0)  {  cerr << "Error: creating thread (id = " << id << ")\n";  return;  }  if (pthread\_detach(thd) != 0)  cerr << "Error: detaching thread (id = " << id << ")\n";  }  ~user()  {  close(pipe\_from);  close(pipe\_to);  pthread\_mutex\_destroy(&mutex);  }  };  struct group  {  string name;  set <string> members;  };  map <int, user\*> users\_id;  map <string, user\*> users;  map <string, group\*> groups;  void\* user\_thd(void\* ptr)  {  user\* usr = (user\*)ptr;  int pipe = usr->pipe\_from;  while (true)  {  query\_id id;  read(pipe, &id, sizeof(query\_id));  switch (id)  {  case SEND\_PRIVATE:  {  message msg = receive\_msg(pipe);  if (users.count(msg.channel) == 0)  {  query\_id ans = SEND\_ERR;  CHECK\_ERROR(pthread\_mutex\_lock(&usr->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(usr->pipe\_to, &ans, sizeof(query\_id)), "Error: answering",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr->mutex), "Error: mutex unlock error\n",);  break;  }  user\* usr\_to = users[msg.channel];  CHECK\_ERROR(pthread\_mutex\_lock(&usr\_to->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(send\_message(usr\_to->pipe\_to, msg, GET\_PRIVATE), "Error: message error\n",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr\_to->mutex), "Error: mutex unlock error\n",);  break;  }  case SEND\_GROUP:  {  message msg = receive\_msg(pipe);  if (groups.count(msg.channel) == 0)  {  query\_id ans = SEND\_ERR;  CHECK\_ERROR(pthread\_mutex\_lock(&usr->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(usr->pipe\_to, &ans, sizeof(query\_id)), "Error: answering",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr->mutex), "Error: mutex unlock error\n",);  break;  }  set <string>& g\_users = groups[msg.channel]->members;  if (g\_users.count(usr->info.username) == 0)  {  query\_id ans = SEND\_ERR;  CHECK\_ERROR(pthread\_mutex\_lock(&usr->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(usr->pipe\_to, &ans, sizeof(query\_id)), "Error: answering",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr->mutex), "Error: mutex unlock error\n",);  break;  }  for (auto it = g\_users.begin(); it != g\_users.end(); it++)  {  if (\*it == msg.author || users.count(\*it) == 0)  continue;  user\* usr\_to = users[\*it];  CHECK\_ERROR(pthread\_mutex\_lock(&usr\_to->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(send\_message(usr\_to->pipe\_to, msg, GET\_GROUP), "Error: message error\n",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr\_to->mutex), "Error: mutex unlock error\n",);  }  break;  }  case CREATE\_GROUP:  {  group\_modify mod = receive\_group\_modify(pipe);  if (groups.count(mod.group) != 0)  {  query\_id ans = CREATE\_G\_ERR;  CHECK\_ERROR(pthread\_mutex\_lock(&usr->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(usr->pipe\_to, &ans, sizeof(query\_id)), "Error: message error\n",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr->mutex), "Error: mutex unlock error\n",);  break;  }  groups[mod.group] = new group();  groups[mod.group]->name = mod.group;  groups[mod.group]->members.insert(usr->info.username);  break;  }  case ADD\_TO\_GROUP:  {  group\_modify mod = receive\_group\_modify(pipe);  if (groups.count(mod.group) == 0 || users.count(mod.username) == 0)  {  query\_id ans = ADD\_G\_ERR;  CHECK\_ERROR(pthread\_mutex\_lock(&usr->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(usr->pipe\_to, &ans, sizeof(query\_id)), "Error: message error\n",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr->mutex), "Error: mutex unlock error\n",);  break;  }  set <string>& g\_users = groups[mod.group]->members;  if (g\_users.count(usr->info.username) == 0 || g\_users.count(mod.username) != 0)  {  query\_id ans = ADD\_G\_ERR;  CHECK\_ERROR(pthread\_mutex\_lock(&usr->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(usr->pipe\_to, &ans, sizeof(query\_id)), "Error: message error\n",);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr->mutex), "Error: mutex unlock error\n",);  break;  }  g\_users.insert(mod.username);  query\_id ans = ADD\_G\_OK;  user\* usr\_to = users[mod.username];  CHECK\_ERROR(pthread\_mutex\_lock(&usr\_to->mutex), "Error: mutex lock error\n", break);  CHECK\_ERROR(write(usr\_to->pipe\_to, &ans, sizeof(query\_id)), "Error: adding in group\n", break);  int n = mod.group.length();  CHECK\_ERROR(write(usr\_to->pipe\_to, &n, sizeof(int)), "Error: adding in group\n", break);  CHECK\_ERROR(write(usr\_to->pipe\_to, mod.group.c\_str(), n), "Error: adding in group\n", break);  CHECK\_ERROR(pthread\_mutex\_unlock(&usr\_to->mutex), "Error: mutex unlock error\n", break);  break;  }  case LOGOUT:  {  string login = usr->info.username;  usr->info.username = "";  users.erase(login);  break;  }  case EXIT:  {  pthread\_mutex\_lock(&usr->mutex);  write(usr->pipe\_to, &id, sizeof(query\_id));  pthread\_mutex\_unlock(&usr->mutex);  delete usr;  return NULL;  }  }  }  return NULL;  }  int main()  {  unlink(data\_pipe.c\_str());  CHECK\_ERROR(mkfifo(data\_pipe.c\_str(), S\_IREAD | S\_IWRITE), "Error: creating data pipe\n", return -1);  sem\_unlink(sem\_name.c\_str());  int data = open(data\_pipe.c\_str(), O\_RDWR);  CHECK\_ERROR(data, "Error: opening data pipe\n", return -1);  sem\_t\* sem = sem\_open(sem\_name.c\_str(), O\_CREAT, S\_IRUSR | S\_IWUSR, 1);  while (true)  {  query\_id id;  read(data, &id, sizeof(query\_id));  switch (id)  {  case LOGIN:  {  int user\_id;  CHECK\_ERROR(read(data, &user\_id, sizeof(int)), "Error: reading user data\n", break);  int len;  CHECK\_ERROR(read(data, &len, sizeof(int)), "Error: reading user data\n", break);  char\* str = (char\*)calloc(len, sizeof(char));  CHECK\_ERROR(read(data, str, len), "Error: reading user data\n", break);  string name = str;  free(str);  query\_id ans;  if (users\_id.count(user\_id) == 0)  users\_id[user\_id] = new user(name, user\_id);  if (users.count(name) == 0)  {  users\_id[user\_id]->info.username = name;  users[name] = users\_id[user\_id];  ans = LOGIN\_OK;  }  else  {  users\_id[user\_id]->info.username = "";  ans = LOGIN\_ERR;  }  CHECK\_ERROR(pthread\_mutex\_lock(&users\_id[user\_id]->mutex), "Error: mutex lock error\n",);  CHECK\_ERROR(write(users\_id[user\_id]->pipe\_to, &ans, sizeof(query\_id)), "Error: answering\n",);  CHECK\_ERROR(pthread\_mutex\_unlock(&users\_id[user\_id]->mutex), "Error: mutex unlock error\n",);  break;  }  case EXIT:  {  int usr\_id;  CHECK\_ERROR(read(data, &usr\_id, sizeof(int)), "Error: reading user id\n", break);  if (users\_id.count(usr\_id) != 0)  {  user\* usr = users\_id[usr\_id];  query\_id ans = EXIT;  write(usr->pipe\_from, &ans, sizeof(query\_id));  }  else  {  int pipe\_to = open((to\_string(usr\_id) + get\_postfix).c\_str(), O\_RDWR);  write(pipe\_to, &id, sizeof(query\_id));  close(pipe\_to);  }  break;  }  }  }  } |

**Демонстрация работы программы**

|  |  |  |
| --- | --- | --- |
| **Терминал 1** | **Терминал 2** | **Терминал 3** |
| liguha@Laptop:~/OS/KP/build$ ./client  login user1  Logged-in  send private user1 Hi, 1 by 1!  Private message from user1: Hi, 1 by 1!  send private user2 Hi, 2 by 1!  send private user3 Hi, 3 by 1!  send private God Are you exists?  Error: can't find receiver  Private message from user2: Hi, 1 by 2!  Private message from user3: Hi, 1 by 3!  chat create 123  chat add 123 user2  chat add 123 user3  chat add 123 noname  Error: can't add this user in this chat  chat add 123 user1  Error: can't add this user in this chat  chat add no\_exist user2  Error: can't add this user in this chat  send chat 123 Hi 2 and 3  Message in chat 123 from user2: Hi 1 and 3  Message in chat 123 from user3: Hi 1 and 2  chat add 23 user1  Error: can't add this user in this chat  chat create 23  Error: can't create such chat  logout  Logged-out  login user2  Logged-in  Message in chat 23 from user3: Who is 2?  Message in chat 123 from user1: Swap 1 and 2!  Private message from user3: Now you are 2  exit  liguha@Laptop:~/OS/KP/build$ | liguha@Laptop:~/OS/KP/build$ ./client  login user2  Logged-in  Private message from user1: Hi, 2 by 1!  send private user1 Hi, 1 by 2!  send private user2 Hi, 2 by 2!  Private message from user2: Hi, 2 by 2!  send private user3 Hi, 3 by 2!  Private message from user3: Hi, 2 by 3!  Welcome in chat 123  Message in chat 123 from user1: Hi 2 and 3  send chat 123 Hi 1 and 3  Message in chat 123 from user3: Hi 1 and 2  chat create 23  chat add 23 user3  send chat 23 Hi 3 and no hi 1  Message in chat 23 from user3: Hi 2 and no hi 1  send private user1 Hi after logout?  Error: can't find receiver  Message in chat 123 from user3: Chat message after logout 1  logout  Logged-out  login user1  Logged-in  send chat 123 Swap 1 and 2!  Private message from user3: Now you are 1  exit  liguha@Laptop:~/OS/KP/build$ | liguha@Laptop:~/OS/KP/build$ ./client  login user1  Error: user already logged-in  login user3  Logged-in  Private message from user1: Hi, 3 by 1!  Private message from user2: Hi, 3 by 2!  send private user1 Hi, 1 by 3!  send private user2 Hi, 2 by 3!  send private user3 Hi, 3 by 3!  Private message from user3: Hi, 3 by 3!  Welcome in chat 123  Message in chat 123 from user1: Hi 2 and 3  Message in chat 123 from user2: Hi 1 and 3  send chat 123 Hi 1 and 2  Welcome in chat 23  Message in chat 23 from user2: Hi 3 and no hi 1  send chat 23 Hi 2 and no hi 1  send chat 123 Chat message after logout 1  send chat 23 Who is 2?  Message in chat 123 from user1: Swap 1 and 2!  send private user1 Now you are 1  send private user2 Now you are 2  logout  Logged-out  send private user1 Try without login  Please, login in the system  chat create new\_chat  Please, login in the system  exit  liguha@Laptop:~/OS/KP/build$ |

**Выводы**

Составлена и отлажена программа на языке C++, реализующая клиент-серверную систему «мгновенных сообщений». Общение между пользователем и сервером осуществляется при помощи пайпов. В системе присутствует возможность создания групповых чатов.