Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Операционные системы»

Студент: А. О. Дубинин
Преподаватель: Е. С. Миронов
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Курсовая работа

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.

Вариант: Клиент и сервер для передачи мнгновенных сообщений. Должен быть один сервер, к которому подключаются пользователи. На сервере должна хранится вся история переписок клиентов. Пользователь может подписаться на "чат"или отписаться от "чата". Сообщение пользователь посылает в конкретный "чат". Если пользователь вышел (или у него случайно закрылось приложение), то сообщение ему должно быть доставлено при его повторном входе.

Транспорт: ZMQ и сигналы

1 Описание

Реализация курсового проекта сводится к следующим задачам:

1. Серверу необходимо уметь индентифицировать клиентов при подключении, причём по уникальному идентификатору процесса.

2. Клиент должен иметь возможность создавать чаты, просматривать список чатов, подключаться и отключаться к чатам.

3. Сервер должен восстанавливать предыдущую сессию максимально точно. Для этого он хранит информацию о чатах, сообщениях, о клиентах и о тех клиентах, кому должно прийти сообщение при их подключении к серверу.

4. Клиент может отправлять сообщение в конкретный чат, если он присоединен к этому чату. Пользователи, которые подключены к чаты, но не онлайн в данный момент, получат сообщения, при повторном подключении к серверу.

Для решения первой задачи, клиенты подключаются к серверу через zmq, передавая идентификатор процесса при включении клиента.

Для решения второй задачи я использовал ZMQ, для общения между клиентом и сервером, через меню клиент общается с сервером, который обрабатывает все команды пользователя.

Восстановление сессии происходит следующим образом. При выключении сервера, все данные сохраняются в файлах и при возобновлении работы, сервер берет данные из файлов.

Для четвертой задачи, пользователь посылает на сервер сообщение и название чата и сервер уже посылает сигнал всем пользователям(кроме отправителя), которые подключены к чату, этот сигнал обрабатывает код клиента и специальной командой через request-reply получает это сообщение.

2 Исходный код

data.h

```
#ifndef DATA_H
#define DATA_H

#include <vector>
#include <string>

typedef enum {
    INIT, LIST_OF_CHATS, CLIENT_OFF,
    ADD_CHAT, JOIN_TO_CHAT, UNSUB,
    SEND_MESSAGE, SEND_TO_SUBS, SHOW_HISTORY
} RequestToken;

typedef struct {
  RequestToken action;
  char clientName[256];
  pid_t client_proc_id;
  char chatName[256];
  char text[256];
  long long status;
} Message ;

typedef struct {
    std::vector<std::string> usersInChat;
    std::string name;
} ChatElem ;


#endif
```

client.cpp

```
#include <zmq.h>
#include <stdio.h>
#include <assert.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/file.h>
#include <unistd.h>
#include <csignal>
#include <sys/types.h>
#include <pthread.h>
#include <iostream>
#include "data.h"

```

```
16 Message message;
17 void *server;
18
19 void menu() {
20     printf("****************************************\n");
21     printf("1 - Show chats \n");
22     printf("2 - Add chat \n");
23     printf("3 - Join to chat \n");
24     printf("4 - Unsubscribe from the chat \n");
25     printf("5 - Send message to chat \n");
26     printf("6 - Show chat's history \n");
27     printf("0 - exit \n");
28     printf("****************************************\n");
29 }
30
31 void send_recv()
32 {
33
34
35     message.action = SEND_TO_SUBS;
36
37     zmq_msg_t request;
38     zmq_msg_init_size(&request, sizeof(Message));
39     memcpy(zmq_msg_data(&request), &message, sizeof(Message));
40     zmq_msg_send(&request, server, 0);
41     zmq_msg_close(&request);
42
43     char ans[256];
44     zmq_msg_init(&request);
45     zmq_msg_recv(&request, server, 0);
46     strcpy(ans, (char *) zmq_msg_data(&request));
47     zmq_msg_close(&request);
48
49     printf("%s",ans);
50 }
51
52 void client_off()
53 {
54
55
56     message.action = CLIENT_OFF;
57
58     zmq_msg_t request;
59     zmq_msg_init_size(&request, sizeof(Message));
60     memcpy(zmq_msg_data(&request), &message, sizeof(Message));
61     zmq_msg_send(&request, server, 0);
62     zmq_msg_close(&request);
63
64     char ans[256];
```

```
65      zmq_msg_init(&request);
66      zmq_msg_recv(&request, server, 0);
67      strcpy(ans, (char *) zmq_msg_data(&request));
68      zmq_msg_close(&request);
69
70      printf("%s\n",ans);
71  }
72
73
74  int main(int argc, char *argv[]) {
75
76
77      std::signal(SIGUSR1,send_recv);
78      std::signal(SIGHUP,client_off);
79
80      int act = 0;
81      char ans[256];
82
83      void *context = zmq_ctx_new();
84      server = zmq_socket(context, ZMQ_REQ);
85
86      ans[0] = '\0';
87      strcat(ans, "tcp://localhost:");
88      if (argc == 2)
89          strcat(ans, argv[1]);
90      else
91          strcat(ans, "4040");
92      int rc = zmq_connect(server, ans);
93      if (rc != 0) {
94          perror("zmq_connect");
95          zmq_close(server);
96          zmq_ctx_destroy(context);
97          exit(1);
98      }
99
100     printf("Enter client name:\n");
101     scanf("%s", message.clientName);
102     message.client_proc_id = getpid();
103     message.action = INIT;
104
105     zmq_msg_t clientReq;
106     zmq_msg_init_size(&clientReq, sizeof(Message));
107     memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
108     zmq_msg_send(&clientReq, server, ZMQ_DONTWAIT);
109     zmq_msg_close(&clientReq);
110
111     zmq_msg_t reply;
112     zmq_msg_init(&reply);
113     zmq_msg_recv(&reply, server, 0);
```

```
114     strcpy(ans, (char *) zmq_msg_data(&reply));
115 
116     if (strcmp(ans, "OK") == 0) {
117         printf("\nWelcome~\n");
118     } else if (strcmp(ans, "ERROR") == 0) {
119         printf("Sorry! Server returned error\n");
120         exit(1);
121     } else {
122         printf("\nWelcome~\n");
123         printf("%s\n", ans);
124     }
125 
126     zmq_msg_close(&reply);
127 
128     for (;;) {
129         menu();
130         if (scanf("%d", &act) == EOF) {
131             act = 0;
132         }
133 
134 
135         if (act == 1) {
136             message.action = LIST_OF_CHATS;
137             zmq_msg_init_size(&clientReq, sizeof(Message));
138             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
139 
140         } else if (act == 2) {
141             message.action = ADD_CHAT;
142             printf("Type chat name:\n");
143             scanf("%s", message.text);
144             zmq_msg_init_size(&clientReq, sizeof(Message));
145             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
146 
147 
148         } else if (act == 3) {
149             message.action = JOIN_TO_CHAT;
150             printf("Type chat name:\n");
151             scanf("%s", message.text);
152             zmq_msg_init_size(&clientReq, sizeof(Message));
153             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
154 
155 
156         } else if (act == 4) {
157             message.action = UNSUB;
158             printf("Type chat name:\n");
159             scanf("%s", message.text);
160             zmq_msg_init_size(&clientReq, sizeof(Message));
161             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
162 
```

```
163
164         } else if (act == 5) {
165             message.action = SEND_MESSAGE;
166             printf("Type chat name:\n");
167             scanf("%s", message.chatName);
168             printf("Type message:\n");
169             getchar();
170             char *line = NULL;
171             size_t size;
172             getline(&line, &size, stdin);
173             strcpy(message.text, line);
174 // scanf("%s", message.text);
175             zmq_msg_init_size(&clientReq, sizeof(Message));
176             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
177 
178 
179         } else if (act == 6) {
180             message.action = SHOW_HISTORY;
181             printf("Type chat name:\n");
182             scanf("%s", message.chatName);
183 
184             zmq_msg_init_size(&clientReq, sizeof(Message));
185             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
186 
187 
188         } else if (act == 0) {
189             message.action = CLIENT_OFF;
190             zmq_msg_init_size(&clientReq, sizeof(Message));
191             memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
192 
193             printf("Sending...\n");
194             zmq_msg_send(&clientReq, server, 0);
195             zmq_msg_close(&clientReq);
196 
197             break;
198 
199         } else {
200             printf("Try more...\n");
201             continue;
202         }
203 
204         printf("Answer:\n");
205         zmq_msg_init_size(&clientReq, sizeof(Message));
206         memcpy(zmq_msg_data(&clientReq), &message, sizeof(Message));
207         zmq_msg_send(&clientReq, server, 0);
208         zmq_msg_close(&clientReq);
209 
210         zmq_msg_init(&reply);
211         zmq_msg_recv(&reply, server, 0);
```

```
212         strcpy(ans, (char *) zmq_msg_data(&reply));
213 
214         if (strcmp(ans, "ERROR") == 0) {
215             printf("Error occured\n");
216         } else {
217             printf("%s\n", ans);
218         }
219 
220         zmq_msg_close(&reply);
221     }
222 
223     zmq_close(server);
224     zmq_ctx_destroy(context);
225     return 0;
226 }
```

server.cpp

```
1  #include "data.h"
2  #include <zmq.h>
3  #include <map>
4  #include <stdio.h>
5  #include <assert.h>
6  #include <unistd.h>
7  #include <string.h>
8  #include <stdlib.h>
9  #include <algorithm>
10 #include <iostream>
11 #include <signal.h>
12 #include <fstream>
13 
14 
15 char ans_publ[256];
16 int END = 0;
17 int NODE = 1;
18 
19 
20 int main(int argc, char *argv[]) {
21 
22     int countOfClients = 0;
23     std::vector<std::pair<std::string, std::string> > history;
24     std::vector<std::pair<std::vector<std::string>, std::string> > debtors; // users,
           message
25     std::map<std::string, pid_t> tree; // map with clients <name, pid>
26     // if pid == 0, client is offline
27     std::vector<ChatElem> chats;
28     Message *message;
29     zmq_msg_t reply;
30     zmq_msg_t request;
31     char ans[256];
```

```
32
33
34      void *context = zmq_ctx_new();
35      void *responder = zmq_socket(context, ZMQ_REP);
36      ans[0] = '\0';
37      strcat(ans, "tcp://*:");
38      char *port;
39      if (argc == 2) {
40          port = argv[1];
41
42      } else {
43          port = (char*)"4040";
44      }
45      strcat(ans, port);
46
47      std::ifstream in("clients.txt", std::ios::in);
48      if (in.is_open()) {
49          while (true) {
50
51              int ctmp;
52              in >> ctmp;
53
54              if (ctmp == END) {
55                  break;
56              }
57
58
59              std::string str;
60              size_t size;
61              in >> size;
62              in >> str;
63
64              pid_t status;
65
66              in >> status;
67              /*std::cout << ctmp << "\n";
68              std::cout << str << "\n";
69              std::cout << status << "\n\n";*/
70              tree[str] = status;
71
72          }
73      }
74
75      std::ifstream in_history("history.txt", std::ios::in);
76      if (in_history.is_open()) {
77          while (true) {
78
79              int ctmp;
80              in_history >> ctmp;
```

```
81
82              if (ctmp == END) {
83                  break;
84              }
85
86
87              std::string name, mes;
88              in_history >> name;
89              std::getline(in_history, mes);
90
91              std::pair<std::string, std::string> chat_message = std::make_pair(name, mes
                    );
92              history.push_back(chat_message);
93
94
95          }
96      }
97
98      std::ifstream in_chats("chats.txt", std::ios::in);
99      if (in_chats.is_open()) {
100         while (true) {
101
102             int ctmp;
103             in_chats >> ctmp;
104             //std::cout << ctmp << "\n";
105
106             if (ctmp == END) {
107                 break;
108             }
109
110             ChatElem elem;
111             in_chats >> elem.name;
112             //std::cout << elem.name << "\n";
113
114             while (true) {
115                 std::string name;
116                 in_chats >> name;
117                 if (name == "-1") {
118                     break;
119                 }
120                 elem.usersInChat.push_back(name);
121             }
122             chats.push_back(elem);
123
124         }
125     }
126
127
128     //std::vector<std::pair<std::vector<std::string>, std::string> > debtors; // users,
```

```
        message
129     std::ifstream in_debtors("debtors.txt", std::ios::in);
130     if (in_debtors.is_open()) {
131         while (true) {
132
133             int ctmp;
134             in_debtors >> ctmp;
135             //std::cout << ctmp << "\n";
136
137             if (ctmp == END) {
138                 break;
139             }
140
141             std::string mes;
142             std::getline(in_debtors, mes);
143             //std::cout << mes << "\n";
144             std::vector<std::string> users;
145             while (true) {
146                 std::string user;
147                 in_debtors >> user;
148                 if (user == "-1") {
149                     break;
150                 }
151                 //std::cout << user << " ";
152                 users.push_back(user);
153             }
154             //std::cout << std::endl;
155             std::pair<std::vector<std::string>, std::string> one_mes = make_pair(users,
                     mes);
156             debtors.push_back(one_mes);
157         }
158     }
159
160     int rc = zmq_bind(responder, ans);
161     if (rc != 0) {
162         perror("zmq_bind");
163         zmq_close(responder);
164         zmq_ctx_destroy(context);
165         exit(1);
166     }
167
168
169     printf("Server initialized\n");
170     for (;;) {
171         zmq_msg_init(&request);
172         zmq_msg_recv(&request, responder, 0);
173         message = (Message *) zmq_msg_data(&request);
174         printf("Recieved message from %s action: %d \n", message->clientName, message->
                 action);
```

```
175         ans[0] = '\0';
176         std::string clientName(message->clientName);
177 
178         std::map<std::string, pid_t>::iterator it;
179         it = tree.find(clientName);
180         if (it == tree.end()) {
181             //register new id
182             printf("Client %s added successfully\n", message->clientName);
183             tree[clientName] = message->client_proc_id;
184         }
185         else if (it->second != message->client_proc_id) {
186             it->second = message->client_proc_id;
187             for(auto &i : debtors) {
188                 //std::vector<std::pair<std::vector<std::string>, std::string> > debtors
                        ; // users, message
189                 for (auto &j : i.first) {
190                     if(clientName == j) {
191 
192                         strcat(ans, i.second.c_str());
193                         strcat(ans, "\n");
194 
195                         std::vector<std::string>::iterator it_vec;
196                         it_vec = find(i.first.begin(), i.first.end(), j);
197                         i.first.erase(it_vec);
198                         break;
199                     }
200                 }
201             }
202         }
203 
204         if (INIT == message->action) {
205             it = tree.find(message->clientName);
206             if (it != tree.end()) {
207                 strcat(ans, "OK");
208                 tree[message->clientName] = message->client_proc_id;
209                 countOfClients++;
210             } else
211                 sprintf(ans, "ERROR");
212         } else if (CLIENT_OFF == message->action) {
213             countOfClients--;
214             tree[message->clientName] = 0;
215             if (countOfClients <= 0) {
216                 printf("all clients are offline, server was down\n");
217                 break;
218             }
219         } else if (LIST_OF_CHATS == message->action) {
220             if (chats.empty()) {
221                 sprintf(ans, "No chats");
222             } else {
```

```
223                 std::string chat = "Chats name(users in chat):\n";
224                 for (auto &i : chats) {
225                     chat += i.name + "(";
226                     for (auto &j : i.usersInChat) {
227                         chat += j + ", ";
228                     }
229                     chat += ")\n";
230 
231                 }
232                 sprintf(ans, chat.c_str());
233             }
234         } else if (ADD_CHAT == message->action) {
235 
236             ChatElem tmp;
237             tmp.name = message->text;
238             chats.push_back(tmp);
239 
240             sprintf(ans, "Chat added");
241         } else if (JOIN_TO_CHAT == message->action) {
242             bool success = false;
243             for (int i = 0; i < chats.size(); ++i) {
244                 if (strcmp(chats[i].name.c_str(), message->text) == 0) {
245                     chats[i].usersInChat.push_back(clientName);
246                     success = true;
247                 }
248             }
249 
250             if (success) {
251 
252                 sprintf(ans, "Client joined to %s", message->text);
253             } else {
254                 sprintf(ans, "error");
255             }
256         } else if (UNSUB == message->action) {
257             bool success = false;
258             for (int i = 0; i < chats.size(); ++i) {
259                 if (strcmp(chats[i].name.c_str(), message->text) == 0) {
260                     std::vector<std::string>::iterator it_vec;
261                     it_vec = find(chats[i].usersInChat.begin(), chats[i].usersInChat.end
                            (), clientName);
262                     if (it_vec != chats[i].usersInChat.end()) {
263                         success = true;
264                         chats[i].usersInChat.erase(it_vec);
265                     } else {
266                         success = false;
267                     }
268 
269                 }
270             }
```

```
271
272             if (success) {
273                 sprintf(ans, "User %s unsub from chat %s", clientName.c_str(), message->
                        text);
274             } else {
275                 sprintf(ans, "error");
276             }
277         } else if (SEND_MESSAGE == message->action) {
278             bool success_chat = false;
279             bool success_user_in_chat = false;
280
281             ans_publ[0] = '\0';
282
283
284             strcat(ans_publ, "(");
285             strcat(ans_publ, message->chatName);
286             strcat(ans_publ, ") ");
287             strcat(ans_publ, message->clientName);
288             strcat(ans_publ, ": ");
289             strcat(ans_publ, message->text);
290
291             std::string chatName(message->chatName);
292             std::string mess(ans_publ);
293             std::pair<std::string, std::string> tmp(chatName, mess);
294             history.push_back(tmp);
295
296
297             for (auto &i : chats) {
298                 if (strcmp(i.name.c_str(), message->chatName) == 0) {
299                     success_chat = true;
300
301                     for (auto &j : i.usersInChat) {
302                         if (j == clientName) {
303                             success_user_in_chat = true;
304                             break;
305                         }
306                     }
307                     if (success_user_in_chat) {
308                         std::vector<std::string> users;
309                         for (auto &j : i.usersInChat) {
310
311                             if (tree[j] == 0) {//std::vector<std::pair<std::vector<std::
                                    string>, std::string> > debtors; // users, message
312                                 users.push_back(j);
313                             }
314
315                             else if (tree[j] != message->client_proc_id) {
316
317                                 kill(tree[j], SIGUSR1);
```

```
318                         }
319                     }
320                     if (!users.empty()) {
321                         std::string chat_message(ans_publ);
322                         std::pair<std::vector<std::string>, std::string> debtor =
                                make_pair(users, chat_message);
323                         debtors.push_back(debtor);
324                     }
325                 }
326             }
327         }
328
329
330         if (success_chat) {
331             if (success_user_in_chat) {
332                 sprintf(ans, "Ok");
333             } else {
334                 sprintf(ans, "User isn't in chat");
335             }
336         } else {
337             sprintf(ans, "Chat isn't exist");
338         }
339
340     } else if (SHOW_HISTORY == message->action) {
341         ans[0] = '\0';
342         for (auto &i : history) {
343             if (strcmp(i.first.c_str(), message->chatName) == 0) {
344
345                 strcat(ans, i.second.c_str());
346                 strcat(ans, "\n");
347             }
348         }
349
350     } else if (SEND_TO_SUBS == message->action) {
351
352         strcat(ans, ans_publ);
353
354
355     } else {
356         sprintf(ans, "ERROR: Wrong request");
357     }
358
359     printf("Send answer to client: [%s]\n", ans);
360     zmq_msg_close(&request);
361     zmq_msg_init_size(&reply, strlen(ans) + 1);
362     memcpy(zmq_msg_data(&reply), ans, strlen(ans) + 1);
363     zmq_msg_send(&reply, responder, 0);
364     zmq_msg_close(&reply);
365 }
```

```
366
367
368     //tree
369     std::ofstream out("clients.txt", std::ios::out);
370     for (auto &kv : tree) {
371         out << NODE << " ";
372         size_t size = kv.first.size();
373         out << size << " ";
374         out.write(&kv.first[0], size);
375         out << " ";
376         out << kv.second;
377         out << "\n";
378     }
379     out << END;
380     out.flush();
381     out.close();
382
383     //std::vector<ChatElem> chats;
384     std::ofstream out_chats("chats.txt", std::ios::out);
385     for (auto &kv : chats) {
386         out_chats << NODE << " ";
387         out_chats << kv.name << " ";
388         for (auto &user : kv.usersInChat) {
389             out_chats << user << " ";
390         }
391         out_chats << "-1";
392         out_chats << "\n";
393     }
394     out_chats << END;
395     out_chats.flush();
396     out_chats.close();
397
398
399     //std::vector<std::pair<std::string, std::string> > history;
400     std::ofstream out_history("history.txt", std::ios::out);
401     for (auto &kv : history) {
402         out_history << NODE << " ";
403         out_history << kv.first << " ";
404         out_history << kv.second << "\n";
405         out_history << "\n";
406     }
407     out_history << END;
408     out_history.flush();
409     out_history.close();
410
411     //std::vector<std::pair<std::vector<std::string>, std::string> > debtors; // users,
          message
412     std::ofstream out_debtors("debtors.txt", std::ios::out);
413     for (auto &kv : debtors) {
```

```
414         out_debtors << NODE << " ";
415         out_debtors << kv.second << "\n";
416         for (auto &user : kv.first) {
417             out_debtors << user << " ";
418         }
419         out_debtors << "-1";
420         out_debtors << "\n";
421     }
422     out_debtors << END;
423     out_debtors.flush();
424     out_debtors.close();
425 
426     zmq_close(responder);
427     zmq_ctx_destroy(context);
428     return 0;
429 }
```

3 Тесты

Меню

```
1 -Show chats
2 -Add chat
3 -Join to chat
4 -Unsubscribe from the chat
5 -Send message to chat
6 -Show chat's history
0 -exit
```

Сервер

```
Server initialized
Recieved message from art action: 0
Send answer to client: [OK]
Recieved message from art action: 1
Send answer to client: [Chats name(users in chat):
vk(max,art,den,)
]
Recieved message from art action: 8
Send answer to client: [ (vk) art: hello world
]
Recieved message from art action: 2
all clients are offline,server was down
```

Клиент art

```
Enter client name:
art

Welcome~
//Присоедние к чату "vk"
3
Type chat name:
vk
Answer:
Client joined to vk
//Просмотр списка чатов
1
Answer:
Chats name(users in chat):
vk(max,art,)
//Послать сообщение в "vk"
5
Type chat name:
vk
Type message:
hello world
Answer:
Ok
//Отключение
0
Sending...
//подключение после рестарта
сервера
Enter client name:
art

Welcome~
//Просмотр списка чатов
1
Answer:
Chats name(users in chat):
vk(max,art,den,)
//Просмотр истории чата
6
Type chat name:
vk
Answer:
(vk) art: hello world
```

Клиент max

```
Enter client name:
max

Welcome~
//Создание чата "vk"
2
Type chat name:
vk
Answer:
Chat added
//Присоедние к чату "vk"
3
Type chat name:
vk
Answer:
Client joined to vk
//Отключение
0
Sending...
//Подключение после того,как
art отправил сообщение в чат
Enter client name:
max

Welcome~
(vk) art: hello world
```

4 Выводы

Выполнив курсовой проект по курсу «Операционные системы», я приобрёл практические навыки в использовании знаний, полученных в течении курса и провел исследование в выбранной предметной области.

Мною был реализован клиент для передачи мгновенныых личных сообщений с помощью zmq и сигналов. В процессе работы на проектом, я понял, что даже такое относительно простое приложение требует множество разносторонних знаний. Созданную мной программу можно дорабатывать ещё очень долго, главная её задача, показать суть использования навыков системного программирования для прикладной программы.