Сети ЭВМ и телекоммуникации

С. А. Климов

24 декабря 2014 г.

Глава 1

Задание

Разработать приложение-клиент и приложение сервер электронной почты.

1.1 Функциональные требования

Серверное приложение должно реализовывать следующие функции:

1. Приём почтового сообщения от одного клиента для другого
2. Хранение электронной почты для клиентов
3. Посылка клиенту почтового сообщения по запросу с последующим удалением сообщения
4. Посылка клиенту сведений о состоянии постового ящика
5. Обработка запроса на отключение клиента
6. Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1. Передача электронного письма на сервер для другого клиента
2. Проверка состояния своего почтового ящика
3. Получение конкретного письма с сервера
4. Разрыв соединения
5. Обработка ситуации отключения клиента сервером

1.2 Нефункциональные требования

Для сервера:

1. Прослушивание определенного порта
2. Поддержка одновременной работы нескольких почтовых клиентов через механизм нитей
3. Обработка запросов на подключение по этому порту от клиентов

Для клиента:

1. Установление соединения с сервером

1.3 Настройки приложений

Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени удалённого сервера почты и номера порта, используемого сервером. Разработанное серверное приложение должно хранить почту для клиентов.

1.4 Накладываемые ограничения

Пакеты для обмена по протоколу TCP будут иметь размер 1024 байта. В пакетах под имя клиента выделяется до 20-ти символов, 1 символ команды, 3 разделяющих симовла, остальные 1000 символов выделяется под отправляемое сообщение. Этой длины достаточно для отправки писем средней длины.

Одновременное количесво подключенных к серверу клиентов как минимум 5.

Глава 2

Реализация для работы по протоколу TCP

2.1 Прикладной протокол

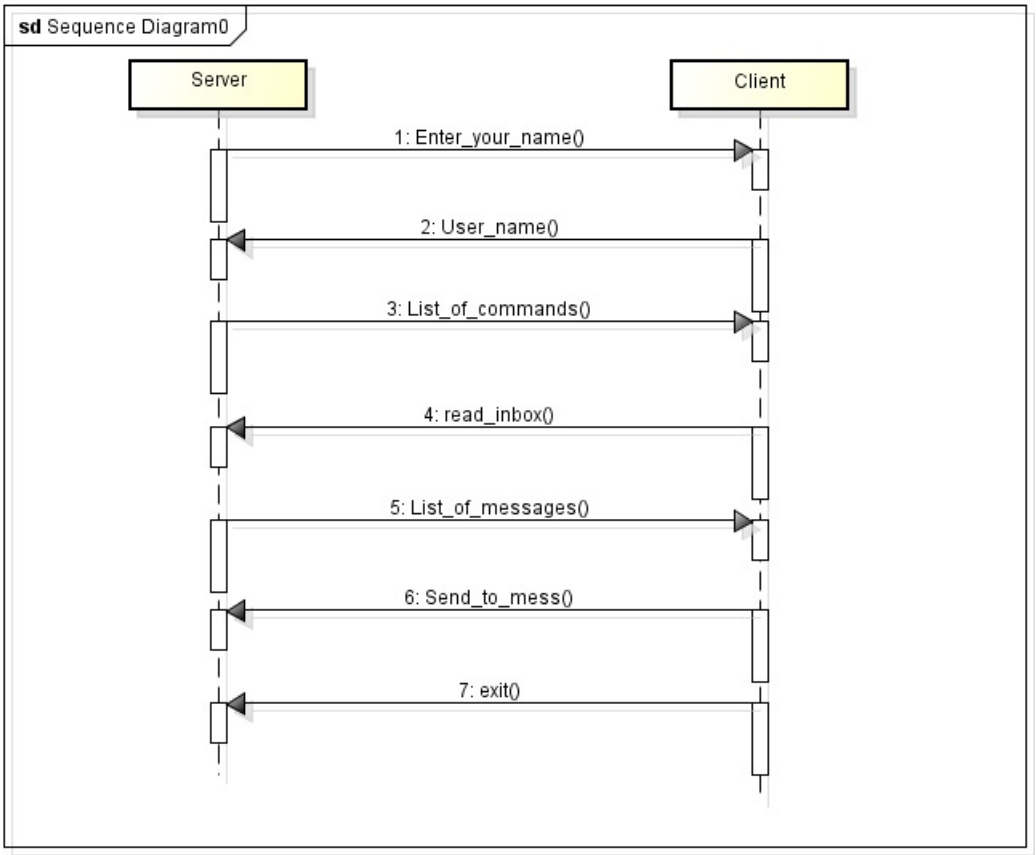
Первая команда ввода имя пользователя	username#
Команда написать kitty сщщбщение hello!	1#kitty#hello!#
Команда прочитать входящие	2#
Команда выхода	3#

2.2 Архитектура приложения

Взаимодействие сервера и клиента:

1. Сервер отправляет приветственное сообщение в котором клиенту предлагается ввести свой логин для идентификации его на сервере.
2. Клиент отправляет серверу свой логин.
3. Сервер отсылает клиенту список поддерживаемых команд(отослать письмо, прочитать входящие письма или выйти).
4. Клиент производит взаимодействие с сервером по средством ввода предложенных команд.
5. Клиент завершает работу с сервером.

Sequence диаграмма, демонстрирующая возможное взаимодействие клиента и сервера(после подключения клиента к серверу):



Почта для каждого клиента хранится на сервере в виде xml файла, имя которого совпадает с именем пользователя.
XML-Schema структуры файлов со входящими сообщениями:

```
1 <xs:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified" xmlns:xs="http://www.w3.org
    /2001/XMLSchema">
2   <xs:element name="inbox">
3       <xs:sequence>
4         <xs:element name="mes" maxOccurs="unbounded"
              minOccurs="0">
5           <xs:simpleContent>
6             <xs:extension base="xs:string">
7               <xs:attribute type="xs:string" name="from"/>
8               <xs:attribute type="xs:string" name="text"/>
9             </xs:extension>
```

```
10              </xs:simpleContent>
11          </xs:element>
12        </xs:sequence>
13    </xs:element>
14 </xs:schema>
```

В данной архитектуре каждому клиенту выделяется свой поток для его обслуживания. Поток начинается при подключении очередного клиента к серверу. Далее он производит взаимодействие с клиентом, обрабатывая отправляемые им команды. Завершение потока происходит при отключении клиента от сервера.

2.3 Тестирование

2.3.1 Описание тестового стенда и методики тестирования

Тестирование проводилось на виртуальной машине Debian 4.7. Было запущено приложение сервера, затем - несколько приложений клента. Таким образом сервер и клиент работали на одном компьютере.

2.3.2 Тестовый план и результаты тестирования

1. На первом клиенте был осуществлен вход под логином "serg"и было отправлнео сообщение "hello!"клиенту "ali"

2. Был получен список входящих сообщений для пользователя "serg".

3. В другом терминале был запущен клиент с именем "ali"(терминал пользователя "serg"оставался активным).

4. Был получен список входящих сообщений, последним из них оказалось только что отправленное сообщение("hello!") пользователем "serg".

5. Было отправлено сообщение "nice to meet you!"пользователю "serg"и осуществлен выход.

6. На терминале пользователя "serg"были прочитаны сообщения, последним из них оказалось только что отправленное сообщение("nice to meet you!") пользователем "ali".

7. Был осуществлен выход пользователя "serg".

Ожидалось получить корректное взаимодействие сервера с клиентами, что и было получено: сервер корректно работает с несколькими клиентами. При вводе некорректных команд клиентское приложение сообщает об этом пользователю и продолжает работу. Серверное приложение так же выводит на консоль некорректные команды от пользователя.

Тестирование работы сервера с большим количесвом клиентов:

1. Был запущен сервер.
2. Из различных терминалов было осуществлено подключение 5 клиентов.
3. Для каждого из клиентов была осуществлена отправка сообщения какому-либо пользователю.
4. В одном из клиентов была добавлена команда sleep(10).
5. Остальные клиенты в это время продолжили взаимодействие с сервером.
6. Для каждого из клиентов был получен список писем.
7. Был осуществлен выход клиентами.

Ожидалось получить корректное взаимодействие сервера с данным количеством клиентов, это и было получено: клиенты корректно отправляют сообщения друг другу и осуществляют просмотр входящих сообщений.

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

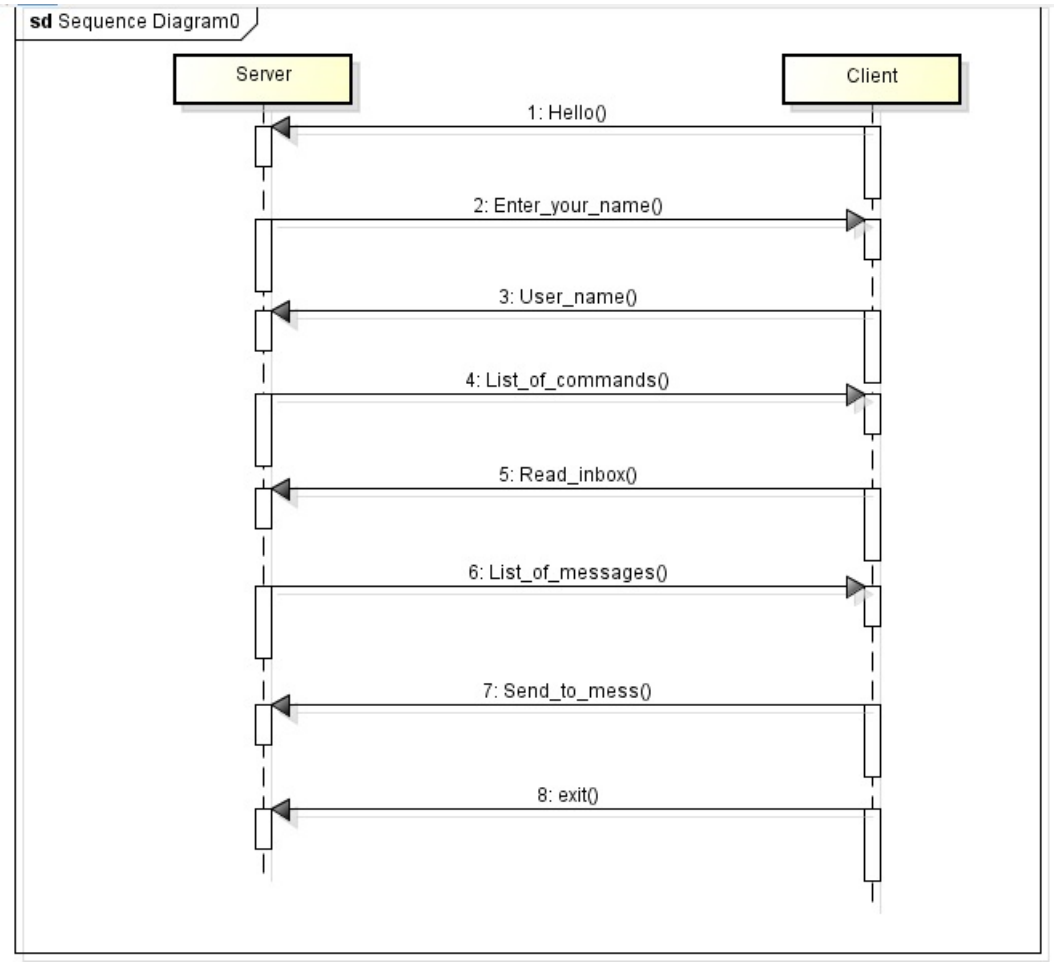
Прикладной протокол понес небольшие изменения по сравнению с UDP. Однако для пользователя взаимодействие с сервером осалось аналогичным.

3.2 Архитектура приложения

Взаимодействие сервера и клиента:

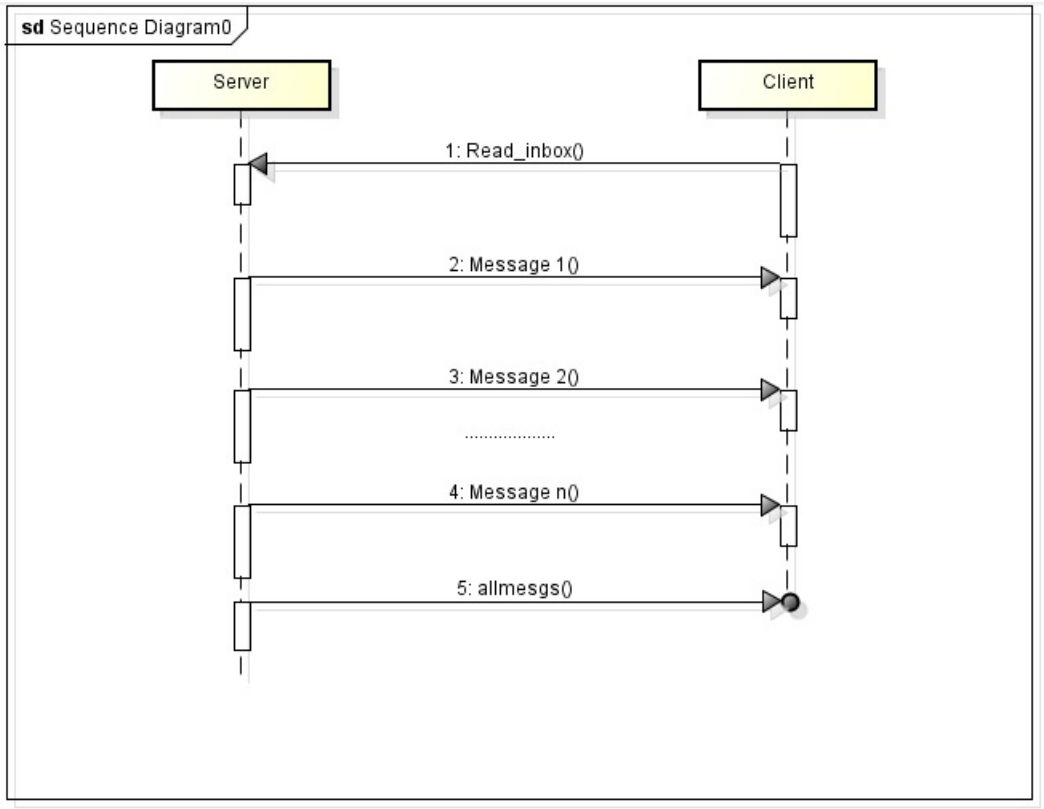
1. Клиент отправляет серверу приветственное сообщение для начала взаимодействия.
2. Сервер отвечает ему сообщением в котором клиенту предлагается ввести свой логин для идентификации его на сервере.
3. Клиент отправляет серверу свой логин.
4. Сервер отсылает клиенту список поддерживаемых команд(отослать письмо, прочитать входящие письма или выйти).
5. Клиент производит взаимодействие с сервером по средством ввода предложенных команд.
6. Клиент завершает работу с сервером.

Sequence диаграмма, демонстрирующая возможное взаимодействие клиента и сервера:



Изменение понес процесс отправки списка всех сообщений клиенту, если в TCP просто осущесвляется потоковая передача, то в UDP мы принимаем пакеты, пока не придет пакет определнного за ранее содержания, сигнализирующий о конце отправки писем.

Sequence диаграмма, демонстрирующая взаимодействие при получении клиентом списка сообщений от сервера:



Почта также хранится в xml файлах, как и в TCP. XML-Schema структуры файлов со входящими сообщениями аналогична.

3.3 Тестирование

3.3.1 Описание тестового стенда и методики тестирования

Тестирование проводилось на виртуальной машине Debian 4.7. Было запущено приложение сервера, затем - несколько приложений клента. Таким образом сервер и клиент работали на одном компьютере.

3.3.2 Тестовый план и результаты тестирования

1. На первом клиенте был осуществлен вход под логином "serg"и было отправлнео сообщение "hello!"клиенту "ali"

2. Был получен список входящих сообщений для пользователя "serg".

3. В другом терминале был запущен клиент с именем "ali"(терминал пользователя "serg"оставался активным).

4. Был получен список входящих сообщений, последним из них оказалось только что отправленное сообщение("hello!") пользователем "serg".

5. Было отправлено сообщение "nice to meet you!"пользователю "serg"и осуществлен выход.

6. На терминале пользователя "serg"были прочитаны сообщения, последним из них оказалось только что отправленное сообщение("nice to meet you!") пользователем "ali".

7. Был осуществлен выход пользователя "serg".

Ожидалось получить корректное взаимодействие сервера с клиентами, что и было получено: сервер корректно работает с несколькими клиентами. При вводе некорректных команд клиентское приложение сообщает об этом пользователю и продолжает работу. Серверное приложение так же выводит на консоль некорректные команды от пользователя.

Тестирование работы сервера с большим количесвом клиентов:

1. Был запущен сервер.

2. Из различных терминалов было осуществлено подключение 5 клиентов.

3. Для каждого из клиентов была осуществлена отправка сообщения какому-либо пользователю.

4. В одном из клиентов была добавлена команда sleep(10).

5. Остальные клиенты в это время не смогли осуществить взаимодействаие с клиентом, они ждали, пока закончится sleep.

6. Для каждого из клиентов был получен список писем.

7. Был осуществлен выход клиентами.

Ожидалось получить корректное взаимодействие сервера с данным количеством клиентов, это и было получено: клиенты корректно отправляют сообщения друг другу и осуществляют просмотр входящих сообщений.

Глава 4

Выводы

4.1 TCP

Протокол TCP является ориентированным на создание соединения, в нем производится так называемое "рукопожатие"для его установки. При установлении соединения становится возможной передача данных в обоих направлениях.
TCP является надежным протоколом, т.к. он управляет подтверждением, повторной передачаей и тайм-аутом сообщений. Также TCP осуществляет контроль порядка доставки сообщений, т.е. если сообщения доставляются не друг за другом, а в случайном порядке, то они внчале кешируются, а затем упорядочиваются и только потом передаются приложению. Важной особенностью TCP является то, что данные считываются как единый поток байтов, нет никаких границ для отдельных сообщений. Очевидным достоинством TCP является его надежность. Однако, при реализации приложния с его использованием можно столкнуться со следующими трудностями: организация работы сервера с несколькими клиентами. Для взаимодействия с несколькими клиентами серверу необходимо создавать новый сокет для каждого из них и передавать управление в отдельную нить, организуемую для обслуживания клиента.

4.2 UDP

Протокол UDP является более простым протоколом, нежели TCP. Он не основан на установлении соединения, а просто и спользует механизм обмена дэйтаграммами для взаимодействия с клиентом.
Протокол UDP считается ненадежным, т.к. он не управляет

подтверждением, повторной передачаей и тайм-аутом сообщений. Он не осуществляет контроль порядка передачи дэйтаграммм. Дэйтаграммы имеют определенные границы, т.е. они интерпретируются на приемной стороне однозначно, их целостность проверяется только после получения. Важной особенностью UDP является его простота - нет механизма "рукопожатия упорядочивания и отслеживания соединений. Хотя UDP является болле простым, на разработчика ложатся дополнительные задачи при его использовании. Необходимо обеспечить контроль очередности доставки пакетов и повторную их отправку при потере пакета. При реализации взаимодействия с несколькими клиентами сервер UDP, в отличии от TCP, использует только один поток для их обслуживания, это необходимо учитывать при работе чтобы отправлять корректные данные тому или иному клиенту. В реализованном сервере используется хранение адреса и порта подключаемого клиента. Это не дает возможности параллельной работы со всеми клиентами, применением мьютексов также не удалось решить данную проблему.
Как показало тестирование, реалиованные приложения как с использованим TCP, так и UDP работают корректно.

Глава 5

Приложения

5.1 Описание среды разработки

Linux Debian 7.6: Среда разработки - Eclipse.
Windows 8.1: Среда разработки - Visual Studio 2013.

5.2 Листинги

5.2.1 Сервер TCP. Основной файл программы main.c

```
1
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7
8  #include <libxml/parser.h>
9  #include <libxml/tree.h>
10 #include <libxml/xmlwriter.h>
11 #include <libxml/xmlmemory.h>
12 #include <libxml/xpath.h>
13
14 #include <sys/stat.h>
15 #include <unistd.h>
16 #include <string.h>
17 #include <pthread.h>
18
19 #define MY_ENCODING "ISO-8859-1"
20 #define MY_PORT 5001
```

```
21
22 void get_args(char* inp_str, char* arg[]){//buffer and array
       to write
23    int i = 0, j;
24    const char s[2] = "#";
25    char *token;
26    token = strtok(inp_str, s);
27    arg[i] = token;
28    while(token != NULL){
29       i++;
30       token = strtok(NULL, s);
31       arg[i] = token;
32    }
33    i--;
34    arg[i] = NULL;
35 }
36
37 char *get_mes_from_client(int newsockfd, char *buffer){
38     int n;
39     bzero(buffer,256);
40     n = read(newsockfd, buffer, 255);
41     if (n < 0)
42     {
43         perror("ERROR reading from socket");
44         exit(1);
45     }
46     return buffer;
47 }
48
49 char* get_hello(int newsockfd, char *buffer, char* arg[]){
50    int i,j=0;
51    char user[10] = "no user";
52    if((arg[0]!=NULL)&&(arg[1]!=NULL)){
53       strcpy(user, arg[0]);
54       char command[7];
55       strcpy(command, arg[1]);
56       j = atoi(command);
57       printf("j=%d\n",j);
58       if (j==1){
59          do_send(user, arg[2], arg[3]);
60       }
61       else if (j==2){
62          do_read(newsockfd, buffer, user);
63       }
64       else if (j==3){
65          do_exit();
66       }
67       else {
68          puts("Wrong command!\n");
```

```
69          }
70      }
71      printf("user: %s\n", user);
72      return user;
73  }
74
75  void do_send(char* from, char* to, char* mes){
76      printf("Send command\n");
77      create_file(to, from, mes);
78  }
79
80  void do_read(int newsockfd, char *buffer, char* client){
81      printf("Read command\n");
82      print_mesgs(buffer, newsockfd, client);
83  }
84
85  void do_exit(){
86      printf("Exit command\n");
87      exit(1);
88  }
89
90  void do_serve(int newsockfd, char *buffer, char* arg[], char*
        user){
91      int i,j=0;
92      if(arg[0]!=NULL){
93          char command[7];
94          strcpy(command, arg[0]);
95          j = atoi(command);
96          printf("j=%d\n",j);
97          if (j==1){
98              do_send(user, arg[1], arg[2]);
99          }
100         else if (j==2){
101             do_read(newsockfd, buffer, user);
102         }
103         else if (j==3){
104             do_exit();
105         }
106         else {
107             puts("Wrong command!\n");
108         }
109     }
110 }
111
112 void do_wait_mes(int newsockfd, char *buffer, char* user){
113     char* inp;
114     char* get_inp[4];
115     while(1){
116         inp=get_mes_from_client(newsockfd, buffer);
```

```
117         get_args(inp, get_inp);
118         do_serve(newsockfd, buffer, get_inp, user);
119     }
120 }
121
122 void add_mes(xmlNode* node, char* from, char* msg){
123     xmlNode* cur_node = NULL;
124     char buf[256];
125     bzero(buf,256);
126     for (cur_node = node; cur_node; cur_node = cur_node->next)
127     {
128         if (cur_node->type == XML_ELEMENT_NODE) {
129             if((!xmlStrcmp(cur_node->name,(const xmlChar *)"
                    inbox"))){
130                 xmlNodePtr nNode = xmlNewNode(0,(const xmlChar
                        *)"mes");
131                 xmlSetProp(nNode,(const xmlChar *)"from",(
                        const xmlChar *)from);
132                 xmlSetProp(nNode,(const xmlChar *)"text",(
                        const xmlChar *)msg);
133                 //xmlNodeSetContent(nNode, (xmlChar*)msg);
134                 xmlAddChild(cur_node,nNode);
135                 return;
136             }
137         }
138         add_mes(cur_node->children, from, msg);
139     }
140 }
141
142 void read_mes(xmlNode* node, char* mess, int newsockfd){
143     xmlNode *cur_node = NULL;
144     char buf[256];
145     bzero(buf,256);
146     int n;
147      for (cur_node = node; cur_node; cur_node = cur_node->next
            ) {
148          if (cur_node->type == XML_ELEMENT_NODE) {
149              if((!xmlStrcmp(cur_node->name,(const xmlChar *)"
                    mes")))
150              {
151                   strcpy(buf,"from:␣");
152                 strncat(buf,xmlGetProp(cur_node,"from"),strlen
                        (xmlGetProp(cur_node,"from")));
153                 strcat(buf,"␣␣␣message:␣");
154                 strncat(buf,xmlGetProp(cur_node,"text"),strlen
                        (xmlGetProp(cur_node,"text")));
155                  strcat(buf,"\n");
156                  n = write(newsockfd,buf,strlen(buf));
157                  //n=read(newsockfd,buf,255);
```

```
158             }
159         }
160
161         read_mes(cur_node->children,mess,newsockfd);
162     }
163 }
164
165 void print_mesgs(char* buffer ,int newsockfd, char* user)
166 {
167
168         xmlDoc         *doc = NULL;
169         xmlNode        *root_element = NULL;
170         int n;
171         doc = xmlReadFile(user, NULL, 0);
172         if (doc == NULL)
173           {
174                   printf("error: could not parse file %s\n",
                            user);
175                   //strncat(buffer,"Error\n",6);
176                   n = write(newsockfd,"Error\n",6);
177           }
178         else
179           {
180
181                   root_element = xmlDocGetRootElement(doc);
182                   n = write(newsockfd,"Messages:\n",11);
183                   read_mes(root_element,buffer,newsockfd);
184                   //n = write(newsockfd,"end",3);
185                   xmlFreeDoc(doc);
186           }
187
188         xmlCleanupParser();
189
190         return;
191
192 }
193
194 void create_file(char *to, char *from, char *msg){
195    int rc;
196    xmlTextWriterPtr writer;
197    xmlDocPtr doc;
198    xmlNodePtr node, root;
199    xmlChar *tmp;
200    if(doc = xmlReadFile(to, NULL, 0)){
201       root = xmlDocGetRootElement(doc);
202       add_mes(root, from, msg);
203       xmlSaveFile(to, doc);
204       xmlFreeDoc(doc);
205    }else{
```

```
206         doc = xmlNewDoc(BAD_CAST XML_DEFAULT_VERSION);
207         node = xmlNewDocNode(doc, NULL, BAD_CAST "inbox", NULL)
                ;
208         xmlDocSetRootElement(doc, node);
209         add_mes(node, from, msg);
210         xmlSaveFile(to, doc);
211         xmlFreeDoc(doc);
212     }
213     xmlCleanupParser();
214 }
215 struct sockParams
216 {
217     int sockfd, newsockfd, port_number, client;
218     struct sockaddr_in serv_addr, cli_addr;
219 };
220
221 void startThread(void *in)
222 {
223     struct sockParams *sp = (struct sockParams *)in;
224     start_work(sp->newsockfd);
225 }
226
227 void start_work(int newsockfd){
228     char buffer[256];
229     int n;
230     bzero(buffer,256);
231         n = read( newsockfd,buffer,255 );
232         if (n < 0)
233         {
234            perror("ERROR␣reading␣from␣socket");
235             exit(1);
236         }
237           //int j;
238             char* get_inp[4];
239             get_args(buffer, get_inp);
240             char user_name[10]="nnm";
241             strcpy(user_name, get_hello(newsockfd, buffer,
                    get_inp));
242             do_wait_mes(newsockfd, buffer, user_name);
243             //return 0;
244 }
245
246 int main( int argc, char *argv[] )
247 {
248     pthread_t thread[5], mainthread;
249     int i=0, j=0;
250     struct sockParams sp;
251
252     sp.sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
253     if (sp.sockfd < 0)
254         {
255             perror("ERROR opening socket");
256             exit(1);
257         }
258     bzero((char *) &sp.serv_addr, sizeof(sp.serv_addr));
259     sp.port_number = MY_PORT;
260     //portno = 7771;
261     sp.serv_addr.sin_family = AF_INET;
262     sp.serv_addr.sin_addr.s_addr = INADDR_ANY;
263     sp.serv_addr.sin_port = htons(sp.port_number);
264 
265         /* Now bind the host address using bind() call.*/
266     if (bind(sp.sockfd, (struct sockaddr *) &sp.serv_addr,
267                              sizeof(sp.serv_addr)) < 0)
268     {
269         perror("ERROR on binding");
270          exit(1);
271     }
272 
273         /* Now start listening for the clients, here process
                will
274         * go in sleep mode and will wait for the incoming
                connection
275         */
276     while(1){
277         listen(sp.sockfd,5);
278         sp.client = sizeof(sp.cli_addr);
279 
280         //printf("cl: %s\n", sp.client);
281       /* Accept actual connection from the client */
282         sp.newsockfd = accept(sp.sockfd, (struct sockaddr *)&sp
                .cli_addr, &sp.client);
283         if (sp.newsockfd < 0)
284         {
285             perror("ERROR on accept");
286             exit(1);
287         }
288         pthread_create(&thread[i], NULL, startThread,(void*)&sp
                );
289         i++;
290     }
291 
292     for(j=0;j<5;j++)
293         pthread_join(thread[j], NULL);
294     return 0;
295 }
```

5.2.2 Сервер TCP. Файл сборки Makefile

```
#it's my makefile
all:
    #gcc -I/home/user/Downloads/libxml2-2.7.8/include main.c -
        Wall -g -O0 -L/home/user/Downloads/libxml2-2.7.8 -lxml2
         -o server1
    gcc -g -lpthread -o0 -Wall -o ./Debug/Server1  main.c -
        lxml2 -I/home/user/Downloads/libxml2-2.7.8/include

clean:
    rm ./Debug/Server1
    rm *.o
```

5.2.3 Клиент TCP. Основной файл программы main.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include <time.h>

void write_socket_start(char* buffer, int sockfd){
    int n;
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    //puts(buffer);
    /* Send message to the server */
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0)
    {
        perror("ERROR writing to socket");
         exit(1);
    }
}

void read_response(char* buffer, int sockfd){
    int n;
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
    {
        perror("ERROR reading from socket");
         exit(1);
    }
```

```
31      printf("%s",buffer);
32  }
33  //-----------------------------------------
34  void get_args(char* inp_str, char* arg[]){//buffer and array
        to write
35      int i = 0, j;
36      const char s[2] = "#";
37      char *token;
38      token = strtok(inp_str, s);
39      arg[i] = token;
40      while(token != NULL){
41          i++;
42          token = strtok(NULL, s);
43          arg[i] = token;
44      }
45      i--;
46      arg[i] = NULL;
47  }
48
49  char *get_mes_from_client(int newsockfd, char *buffer){
50       int n;
51       bzero(buffer,256);
52       n = read(newsockfd, buffer, 255);
53       if (n < 0)
54       {
55            perror("ERROR reading from socket");
56            exit(1);
57       }
58       return buffer;
59  }
60
61  void resp_send(char* from, char* to, char* mes){
62  }
63
64  void resp_read(int newsockfd, char *buffer, char* client){
65      print_mesgs(buffer, newsockfd, client);
66  }
67
68  void resp_exit(){
69      exit(1);
70  }
71
72  void print_mesgs(char* buffer ,int sockfd, char* user){
73      read_response(buffer, sockfd);
74      read_response(buffer, sockfd);
75  }
76
77  char* put_hello(int sockfd, char *buffer){
78      write_socket_start(buffer, sockfd);
```

```
79      char* arg[4];
80      get_args(buffer, arg);
81      int i,j=0;
82      char user[10] = "no user";
83      if((arg[0]!=NULL)&&(arg[1]!=NULL)){
84          strcpy(user, arg[0]);
85          char command[7];
86          strcpy(command, arg[1]);
87          j = atoi(command);
88          if (j==1){
89              resp_send(user, arg[2], arg[3]);
90          }
91          else if (j==2){
92              resp_read(sockfd, buffer, user);
93          }
94          else if (j==3){
95              resp_exit();
96          }
97          else {
98              puts("Wrong command!\n");
99          }
100     }
101     return user;
102 }
103
104 void keep_talking(int sockfd, char *buffer){
105     write_socket_start(buffer, sockfd);
106         char* arg[4];
107         get_args(buffer, arg);
108         int i,j=0;
109         char user[10] = "no user";
110         if((arg[0]!=NULL)){
111             strcpy(user, arg[0]);
112             char command[7];
113             strcpy(command, arg[0]);
114             j = atoi(command);
115             if (j==1){
116                 resp_send(user, arg[2], arg[3]);
117             }
118             else if (j==2){
119                 resp_read(sockfd, buffer, user);
120             }
121             else if (j==3){
122                 resp_exit();
123             }
124             else {
125                 puts("Wrong command!\n");
126             }
127         }
```

```
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];

    if (argc < 3) {
        fprintf(stderr,"usage %s hostname port\n", argv[0]);
        exit(0);
    }
    portno = atoi(argv[2]);
    /* Create a socket point */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("ERROR opening socket");
        exit(1);
    }
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(portno);

    /* Now connect to the server */
    if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
    {
         perror("ERROR connecting");
         exit(1);
    }
    /* Now ask for a message from the user, this message
    * will be read by server
    */
    put_hello(sockfd, buffer);

    while(1)
      keep_talking(sockfd, buffer);
    return 0;
}
```

5.2.4 Клиент TCP. Файл сборки Makefile

```
#it's my makefile
all:
   #gcc -I/home/user/Downloads/libxml2-2.7.8/include main.c -
      Wall -g -O0 -L/home/user/Downloads/libxml2-2.7.8 -lxml2
       -o server1
   gcc -g -o0 -Wall -o ./Debug/Client1  main.c -lxml2 -I/home
      /user/Downloads/libxml2-2.7.8/include

clean:
   rm ./Debug/Client1
   rm *.o
```

5.2.5 Сервер UDP. Основной файл программы main.c

```
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>
#include <assert.h>

#include <libxml/parser.h>
#include <libxml/tree.h>
#include <libxml/xmlwriter.h>
#include <libxml/xmlmemory.h>
#include <libxml/xpath.h>

#define BUFLEN 1024  //Max length of buffer
#define MY_PORT 5001   //The port on which to listen for
   incoming data

void die(char *s)
{
    perror(s);
    exit(1);
}

char** splito_array(char* str){
   //char    str[]= "ls -l";
   char ** res  = NULL;
   char *  p    = strtok (str, "#");
   int n_spaces = 0, i;


```

```
30    /* split string and append tokens to 'res' */
31
32    while (p) {
33      res = realloc (res, sizeof (char*) * ++n_spaces);
34
35      if (res == NULL)
36        exit (-1); /* memory allocation failed */
37
38      res[n_spaces-1] = p;
39
40      p = strtok (NULL, "#");
41    }
42
43    /* realloc one extra element for the last NULL */
44
45    res = realloc (res, sizeof (char*) * (n_spaces+1));
46    res[n_spaces] = 0;
47
48    /* free the memory allocated */
49
50    free (res);
51    return res;
52  }
53
54  void get_args(char* inp_str, char* arg[]){//buffer and array
      to write
55    int i = 0, j;
56    const char s[2] = "#";
57    char *token;
58    token = strtok(inp_str, s);
59    arg[i] = token;
60    while(token != NULL){
61       i++;
62       token = strtok(NULL, s);
63       arg[i] = token;
64    }
65    i--;
66    arg[i] = NULL;
67  }
68
69  char** str_split(char* a_str, const char a_delim)
70  {
71      char** result    = 0;
72      size_t count     = 0;
73      char* tmp        = a_str;
74      char* last_comma = 0;
75      char delim[2];
76      delim[0] = a_delim;
77      delim[1] = 0;
```

```
78
79      /* Count how many elements will be extracted. */
80      while (*tmp)
81      {
82          if (a_delim == *tmp)
83          {
84              count++;
85              last_comma = tmp;
86          }
87          tmp++;
88      }
89
90      /* Add space for trailing token. */
91      count += last_comma < (a_str + strlen(a_str) - 1);
92
93      /* Add space for terminating null string so caller
94         knows where the list of returned strings ends. */
95      count++;
96
97      result = malloc(sizeof(char*) * count);
98
99      if (result)
100     {
101         size_t idx  = 0;
102         char* token = strtok(a_str, delim);
103
104         while (token)
105         {
106             assert(idx < count);
107             *(result + idx++) = strdup(token);
108             token = strtok(0, delim);
109         }
110         assert(idx == count - 1);
111         *(result + idx) = 0;
112     }
113
114     return result;
115 }
116
117 void do_send(char* from, char* to, char* mes){
118    printf("Send␣command\n");
119    create_file(to, from, mes);
120 }
121
122 void do_read(int newsockfd, char *buffer, char* client,
       struct sockaddr_in si_other, int slen){
123    printf("Read␣command\n");
124    print_mesgs(buffer, newsockfd, client, si_other, slen);
125 }
```

```
126
127 void read_mes(xmlNode* node, char* mess, int s, struct
       sockaddr_in si_other, int slen){
128    xmlNode *cur_node = NULL;
129    char buf[BUFLEN];
130    bzero(buf,BUFLEN);
131    int n;
132     for (cur_node = node; cur_node; cur_node = cur_node->next
          ) {
133         if (cur_node->type == XML_ELEMENT_NODE) {
134             if((!xmlStrcmp(cur_node->name,(const xmlChar *)"
                   mes")))
135             {
136                  strcpy(buf,"from:␣");
137                 strncat(buf,xmlGetProp(cur_node,"from"),strlen
                       (xmlGetProp(cur_node,"from")));
138                 strcat(buf,"␣␣␣message:␣");
139                 strncat(buf,xmlGetProp(cur_node,"text"),strlen
                       (xmlGetProp(cur_node,"text")));
140                  strcat(buf,"\n");
141
142                  if (sendto(s, buf, sizeof(buf),
143                                                          0 , (
                                                              struct
                                                              sockaddr
                                                               *) &
                                                              si_other
                                                              , slen)
                                                              ==-1)
144                                                          {
145                                                              die
                                                                  (
                                                                  "
                                                                  sendto
                                                                  ()
                                                                  "
                                                                  )
                                                                  ;
146                                                          }
147
148                                                          //
                                                              receive
                                                               a
                                                              reply
                                                               and

                                                              print
                                                               it
```

```
149                                                 //clear the buffer by filling null, it might have previously received data
150                                                 //memset(buf,'\0', BUFLEN);
151                                                 //try to receive some data, this is a blocking call
152                                                 if (recvfrom(s, buf, sizeof(buf),
```

```
153                                                                 0,
                                                                    (
                                                                    struct
                                                                    sockaddr
                                                                    *)
                                                                    &
                                                                    si_other
                                                                    ,
                                                                    &
                                                                    slen
                                                                    )
                                                                    ==
                                                                    -1)
154                                                             {
155                                                                 //
                                                                        puts
                                                                        ("
                                                                        qq
                                                                        ")
                                                                        ;
156                                                                  //
                                                                        die
                                                                        ("
                                                                        recvfrom
                                                                        ()
                                                                        ")
                                                                        ;
157                                                             }
158
159                 }
160             }
161
162             read_mes(cur_node->children,mess,s, si_other, slen);
163         }
164 }
165
166 void print_mesgs(char* buffer ,int s, char* user, struct
        sockaddr_in si_other, int slen)
167 {
168
```

```
169         xmlDoc          *doc = NULL;
170         xmlNode         *root_element = NULL;
171         int n;
172         doc = xmlReadFile(user, NULL, 0);
173         if (doc == NULL)
174           {
175                   printf("error: could not parse file %s\n",
                            user);
176           }
177         else
178           {
179
180                   root_element = xmlDocGetRootElement(doc);
181
182                   if (sendto(s, "Messages:\n", sizeof("
                            Messages:\n"),
183                                   0 , (struct sockaddr *) &
                                        si_other, slen)==-1)
184                                   {
185                                       die("sendto()");
186                                   }
187
188                                   //receive a reply and
                                        print it
189                                   //clear the buffer by
                                        filling null, it
                                        might have
                                        previously received
                                        data
190                                   //memset(buf,'\0',
                                        BUFLEN);
191                                   //try to receive some
                                        data, this is a
                                        blocking call
192                                   if (recvfrom(s, "
                                        Messages:\n", sizeof
                                        ("Messages:\n"),
193                                        0, (struct sockaddr
                                             *) &si_other, &
                                            slen) == -1)
194                                   {
195                                    //puts("qq");
196                                       //die("recvfrom()")
                                            ;
197                                   }
198
199
200                   //n = write(newsockfd,"Messages:\n",11);
```

```
201                 read_mes(root_element,buffer,s, si_other,
                        slen);
202                 //n = write(newsockfd,"end",3);
203
204                 if (sendto(s, "allmesgs", sizeof("allmesgs"
                        ),
205                                                     0 , (
                                                         struct
                                                         sockaddr
                                                          *)
                                                         &
                                                         si_other
                                                         ,
                                                         slen
                                                         )
                                                         ==-1)
206                                                     {
207                                                         die
                                                                (
                                                                "
                                                                sendto
                                                                ()
                                                                "
                                                                )
                                                                ;
208                                                     }
209
210                                                     //
                                                            receive
                                                             a
                                                            reply
                                                            and
                                                            print
                                                            it
211                                                     //
                                                            clear
                                                            the
                                                            buffer
                                                            by
```

```
                                                    filling
                                                    null
                                                    ,
                                                    it
                                                    might
                                                    have
                                                    previously
                                                    received
                                                    data
212                                                 //
                                                    memset
                                                    (
                                                    buf
                                                    ,'\0',
                                                    BUFLEN
                                                    );
213
214                 if (recvfrom(s, "allmesgs", sizeof("
                        allmesgs"),
215                                                                                 C
```

```
216                                                                 {
217                                                                     //
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
218                                                                     //
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
219                                                                 }
220
221                     xmlFreeDoc(doc);
222             }
223
224         xmlCleanupParser();
225
226         return;
227
228 }
229
230 void add_mes(xmlNode* node, char* from, char* msg){
231     xmlNode* cur_node = NULL;
232     char buf[BUFLEN];
233     bzero(buf,BUFLEN);
234     for (cur_node = node; cur_node; cur_node = cur_node->next)
235     {
236         if (cur_node->type == XML_ELEMENT_NODE) {
237             if((!xmlStrcmp(cur_node->name,(const xmlChar *)"
                    inbox"))){
238                 xmlNodePtr nNode = xmlNewNode(0,(const xmlChar
                        *)"mes");
239                 xmlSetProp(nNode,(const xmlChar *)"from",(
                        const xmlChar *)from);
240                 xmlSetProp(nNode,(const xmlChar *)"text",(
                        const xmlChar *)msg);
241                 //xmlNodeSetContent(nNode, (xmlChar*)msg);
242                 xmlAddChild(cur_node,nNode);
243                 return;
244             }
245         }
```

```
246         add_mes(cur_node->children, from, msg);
247     }
248 }
249 
250 void create_file(char *to, char *from, char *msg){
251     int rc;
252     xmlTextWriterPtr writer;
253     xmlDocPtr doc;
254     xmlNodePtr node, root;
255     xmlChar *tmp;
256     if(doc = xmlReadFile(to, NULL, 0)){
257         root = xmlDocGetRootElement(doc);
258         add_mes(root, from, msg);
259         xmlSaveFile(to, doc);
260         xmlFreeDoc(doc);
261     }else{
262         doc = xmlNewDoc(BAD_CAST XML_DEFAULT_VERSION);
263         node = xmlNewDocNode(doc, NULL, BAD_CAST "inbox", NULL)
                ;
264         xmlDocSetRootElement(doc, node);
265         add_mes(node, from, msg);
266         xmlSaveFile(to, doc);
267         xmlFreeDoc(doc);
268     }
269     xmlCleanupParser();
270 }
271 
272 int get_cmd(char* buffer, int num){
273     int cmd;
274     char **temp;
275     temp=str_split(buffer, '#');
276 
277             if(temp){
278                 int i;
279                  for (i = 0; *(temp + i); i++)
280                  {
281                    printf("inp␣:%s\n", *(temp + i));
282                      if(i == num)
283                       cmd = atoi(*(temp + i));
284                      free(*(temp + i));
285                  }
286                  free(temp);
287             }
288     return cmd;
289 }
290 
291 char* get_cmd_ch(char* buffer, int num, char* res){
292     char* cmd;
293     char **temp;
```

```
294     char *teeemp;
295     temp=str_split(buffer, '#');
296 
297             if(temp){
298                int i;
299                for (i = 0; *(temp + i); i++)
300                {
301 
302                     if(i == num)
303                      cmd = (*(temp + i));
304 
305                     strcpy(res, (*(temp + i)));
306                     free(*(temp + i));
307                }
308                free(temp);
309             }
310     return cmd;
311 }
312 
313 int use_token(char* str, int num, char* res){
314     const char s[2] = "#";
315     char *token;
316     int i = 0;
317 
318     token = strtok(str, s);
319     if(num==0){
320        res = token;
321        return 0;
322     }
323     while(token != NULL){
324        printf("i:%d, t:%s", i, token);
325        i++;
326        token = strtok(NULL, s);
327        if(num==i){
328           res = token;
329           return 0;
330        }
331     }
332     return 0;
333 }
334 
335 void squeeze (char s[], int c) {
336     int i, j;
337 
338     for (i = j = 0; s[i] != '\0'; i++)
339        if (s[i] != c)
340           s[j++] = s[i];
341     s[j] = '\0';
342 }
```

```
343
344 int main(void)
345 {
346     struct sockaddr_in si_me, si_other;
347
348     int s, i, slen = sizeof(si_other) , recv_len;
349     char buf[BUFLEN];
350
351     char* user_names[5];
352     char* user_addr[5];
353     //int user_port[5];
354     uint16_t user_port[5];
355
356     //create a UDP socket
357     if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
358     {
359         die("socket");
360     }
361
362     // zero out the structure
363     memset((char *) &si_me, 0, sizeof(si_me));
364
365     si_me.sin_family = AF_INET;
366     si_me.sin_port = htons(MY_PORT);
367     si_me.sin_addr.s_addr = htonl(INADDR_ANY);
368
369     //bind socket to port
370     if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) ==
            -1)
371     {
372         die("bind");
373     }
374
375     //keep listening for data
376
377 socklen_t slen_o = sizeof(si_other);
378 int sa = sizeof("Commands:␣1#kitty#Hello#␣-␣Send␣message␣'
    Hello'␣to␣user␣kitty;\n"
379       "2#␣-␣read␣ur␣inbox;␣3#␣-␣exit\n");
380 int k=0;
381
382 char* arg[4];
383 char* arg_tmp[4];
384 //char user_name[10]="nnm";
385 char* user_name;
386 //char* send_to;
387 //char* mess_to;
388 char buf_tmp[BUFLEN];
389 //char buf_tmp2[BUFLEN];
```

```
390 //char buf_tmp3[BUFLEN];
391 char users[5][20];
392 char adrs[5][30];
393
394 char** teem;
395
396 int arr[2] = {0,1};
397
398 //number = 0;
399
400
401 char **qq;
402
403     while(1)
404     {
405       bzero(buf, sizeof(buf));
406         //printf("Waiting for data...");
407         fflush(stdout);
408
409         //try to receive some data, this is a blocking call
410         if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct
              sockaddr *) &si_other, &slen_o)) == -1)
411         {
412             die("recvfrom()");
413         }
414
415
416
417         //now reply the client with the same data
418         if (sendto(s, buf, recv_len, 0, (struct sockaddr*) &
              si_other, slen_o) == -1)
419         {
420             die("sendto()");
421         }
422
423         //if(nm < 5){
424          //user_addr[nm] =
425         //}
426
427
428         printf("Received␣packet␣from␣%s:%d\n", inet_ntoa(
              si_other.sin_addr), ntohs(si_other.sin_port));
429
430         strcpy(buf_tmp,buf);
431         int buu = 0;
432         buu = get_cmd(buf,0);
433         //printf("b = %d", buu);
434
435         if(strcmp(buf, "Hello") == 0){
```

```
436             if (sendto(s, "Enter␣ur␣name␣ending␣with␣'#'\n",
                    sizeof("Enter␣ur␣name␣ending␣with␣'#'\n"),
437                     0 , (struct sockaddr *) &si_other, slen_o)
                        ==-1)
438                 {
439                     die("sendto()");
440                 }
441
442                 //receive a reply and print it
443                 //clear the buffer by filling null, it
                        might have previously received data
444                 //memset(buf,'\0', BUFLEN);
445                 //try to receive some data, this is a
                        blocking call
446                 if (recvfrom(s, "Enter␣ur␣name␣ending␣
                    with␣'#'\n", sizeof("Enter␣ur␣name␣
                    ending␣with␣'#'\n"),
447                     0, (struct sockaddr *) &si_other, &
                        slen_o) == -1)
448                 {
449                  //puts("qq");
450                     //die("recvfrom()");
451                 }
452                 if ((recv_len = recvfrom(s, buf, BUFLEN,
                    0, (struct sockaddr *) &si_other, &
                    slen_o)) == -1)
453                     {
454                         die("recvfrom()");
455                     }
456
457                     //print details of the client/
                        peer and the data received
458                     //printf("Received packet from %s
                        :%d\n", inet_ntoa(si_other.
                        sin_addr), ntohs(si_other.
                        sin_port));
459                     //printf("Data: %s\n" , buf);
460
461                     //now reply the client with the
                        same data
462                     if (sendto(s, buf, recv_len, 0, (
                        struct sockaddr*) &si_other,
                        slen_o) == -1)
463                     {
464                         die("sendto()");
465                     }
466
467
```

```
468                             //bzero(user_name, sizeof(
                                    user_name));
469
470                             //teem = splito_array(buf);
471                             //printf("us: %s", teem[0]);
472                             //strcpy(user_name, teem[0]);
473                             //get_cmd_ch(buf, 0, user_name);
474
475                             strcpy(user_name, buf);
476                             //strcpy(user_names[arr[0]], buf)
                                    ;
477                             user_name[strlen(user_name)-1]=0;
478                             //user_names[arr[0]][strlen(
                                    user_names[arr[0]])-1]=0;
479
480                             strcpy(users[arr[0]], buf);
481
482
483                             squeeze(users[arr[0]], '#');
484
485                             strcpy(adrs[arr[0]], inet_ntoa(
                                    si_other.sin_addr));
486
487                             user_port[arr[0]] = ntohs(
                                    si_other.sin_port);
488
489                             arr[0]++;
490
491
492
493                             printf("User: %s\n", user_name);
494                             bzero(buf, sizeof(buf));
495         }
496         else if(strcmp(buf, "Start") == 0){
497          if (sendto(s, "Commands: 1# kitty# Hello# - Send
                message 'Hello' to user kitty;\n"
498                           "2# - read ur inbox; 3# - exit\n",
                                  sa , 0 , (struct sockaddr *) &
                                 si_other, slen_o)==-1)
499                            {
500                                die("sendto()");
501                            }
502
503                            //receive a reply and print it
504                            //clear the buffer by filling
                                  null, it might have
                                  previously received data
505                            //memset(buf,'\0', BUFLEN);
```

```
506                             //try to receive some data, this
                                    is a blocking call
507                             if (recvfrom(s, "Commands:␣1#
                                   kitty#Hello#␣-␣Send␣message␣'
                                   Hello'␣to␣user␣kitty;\n"
508                              "2#␣-␣read␣ur␣inbox;␣3#␣-␣exit\
                                   n", sa, 0, (struct sockaddr
                                   *) &si_other, &slen_o) ==
                                   -1)
509                             {
510                              //puts("qq");
511                                 //die("recvfrom()");
512                             }
513                             bzero(buf, sizeof(buf));
514
515         }
516         else if(buu < 4){
517
518          puts("OBRABOTKA\n");
519
520          printf("usss:␣%s",users[0]);
521
522          qq = splito_array(buf_tmp);
523
524          if(buu!=0){
525             if (buu==1){
526                //printf("name: %s, to: %s, mes: %s\n",
                      user_name, qq[1], qq[2]);
527                for(k=0;k<5;k++)
528                   if(user_port[k] == ntohs(si_other.sin_port)
                         ){
529                      printf("k=%d␣p:␣%d,␣us:␣%s\n", k,
                         user_port[k], users[k]);
530                      //do_send(user_names[k], qq[1], qq[2]);
531                      printf("from=%s␣to:␣%s,␣mes:␣%s\n",
                         users[k], qq[1], qq[2]);
532                      do_send(users[k], qq[1], qq[2]);
533                   }
534
535                //for(k=0;k<5;k++)
536                   //if(strcmp(user_port[k], ntohs(si_other.
                         sin_port)) == 0)
537                   //strcpy(user_name, user_names[k]);
538
539                //do_send(user_name, qq[1], qq[2]);
540
541                //puts("com 1");
542             }
543             else if (buu==2){
```

```
544                 for(k=0;k<5;k++)
545                     if(user_port[k] == ntohs(si_other.sin_port)
                            ){
546                         printf("Client name: %s\n", user_name);
547                         //do_read(s, buf, user_name, si_other,
                                slen);
548                         do_read(s, buf, users[k], si_other, slen
                                );
549                     }
550                 //puts("com 2");
551             }
552             else if (buu==3){
553                 /*if(strcmp(user,"root") == 0){
554                     do_exit(user);
555                     exit(1);
556                 }
557                 else
558                     do_exit(user);*/
559                 //puts("com 3");
560             }
561             else {
562                 puts("Wrong command!\n");
563             }
564          }
565
566
567          //printf("Data: %s\n" , buf);
568          bzero(buf, sizeof(buf));
569         }
570         bzero(buf, sizeof(buf));
571     }
572
573     close(s);
574     return 0;
575 }
```

5.2.6 Сервер UDP. Файл сборки Makefile

```
1 #it's my makefile
2 all:
3    #gcc -I/home/user/Downloads/libxml2-2.7.8/include main.c -
        Wall -g -O0 -L/home/user/Downloads/libxml2-2.7.8 -lxml2
         -o server1
4    gcc -g -o0 -Wall -o ./Debug/Server3  main.c -lxml2 -I/home
        /user/Downloads/libxml2-2.7.8/include
5
6 clean:
```

```
7     rm ./Debug/Server3
8     rm *.o
```

5.2.7 Клиент UDP. Основной файл программы main.c

```
1  #include<stdio.h> //printf
2  #include<string.h> //memset
3  #include<stdlib.h> //exit(0);
4  #include<arpa/inet.h>
5  #include<sys/socket.h>
6  #include <unistd.h>
7  #include <assert.h>
8
9  #define SERVER "127.0.0.1"
10 #define BUFLEN 512  //Max length of buffer
11 #define PORT 5001   //The port on which to send data
12
13 void die(char *s)
14 {
15     perror(s);
16     exit(1);
17 }
18
19 char** str_split(char* a_str, const char a_delim)
20 {
21     char** result    = 0;
22     size_t count     = 0;
23     char* tmp        = a_str;
24     char* last_comma = 0;
25     char delim[2];
26     delim[0] = a_delim;
27     delim[1] = 0;
28
29     /* Count how many elements will be extracted. */
30     while (*tmp)
31     {
32         if (a_delim == *tmp)
33         {
34             count++;
35             last_comma = tmp;
36         }
37         tmp++;
38     }
39
40     /* Add space for trailing token. */
41     count += last_comma < (a_str + strlen(a_str) - 1);
```

```
42
43      /* Add space for terminating null string so caller
44         knows where the list of returned strings ends. */
45      count++;
46
47      result = malloc(sizeof(char*) * count);
48
49      if (result)
50      {
51          size_t idx  = 0;
52          char* token = strtok(a_str, delim);
53
54          while (token)
55          {
56              assert(idx < count);
57              *(result + idx++) = strdup(token);
58              token = strtok(0, delim);
59          }
60          assert(idx == count - 1);
61          *(result + idx) = 0;
62      }
63
64      return result;
65  }
66
67  char** splito_array(char* str){
68     //char    str[]= "ls -l";
69     char ** res  = NULL;
70     char *  p    = strtok (str, "#");
71     int n_spaces = 0, i;
72
73
74     /* split string and append tokens to 'res' */
75
76     while (p) {
77       res = realloc (res, sizeof (char*) * ++n_spaces);
78
79       if (res == NULL)
80         exit (-1); /* memory allocation failed */
81
82       res[n_spaces-1] = p;
83
84       p = strtok (NULL, "#");
85     }
86
87     /* realloc one extra element for the last NULL */
88
89     res = realloc (res, sizeof (char*) * (n_spaces+1));
90     res[n_spaces] = 0;
```

```
91
92      /* print the result */
93
94      //for (i = 0; i < (n_spaces+1); ++i)
95        //printf ("res[%d] = %s\n", i, res[i]);
96
97      /* free the memory allocated */
98
99      free (res);
100     return res;
101 }
102
103 int get_cmd(char* buffer, int num){
104     int cmd;
105     char **temp;
106     char* arr1;
107     char* arr2;
108     temp=str_split(buffer, '#');
109
110             if(temp){
111                 int i;
112                  for (i = 0; *(temp + i); i++)
113                  {
114                    //printf("inp :%s\n", *(temp + i));
115                      if(i == num)
116                       cmd = atoi(*(temp + i));
117                      else if(i == 1)
118                       //memcpy(arr1, (*(temp + i)), strlen(*(
                              temp + i))+1);
119                       //strcpy(arr1, (**(temp + i)));
120                      //else if(i == 2)
121                       //strcpy(arr2, (*(temp + i)));
122                      free(*(temp + i));
123                  }
124                    //printf("\n");
125                  free(temp);
126             }
127     //printf("After: %s\n", arr1);
128     //printf("After: %s\n", arr2);
129     return cmd;
130 }
131
132 void get_cmd_ch(char* buffer, int num, char* res){
133     char* cmd;
134     char **temp;
135     temp=str_split(buffer, '#');
136
137             if(temp){
138                 int i;
```

```
139                 for (i = 0; *(temp + i); i++)
140                 {
141                   printf("inp_ch␣:%s\n", *(temp + i));
142                   //printf("i= %d inp :%s\n", i, teeemp);
143 
144                      //cmd = (*(temp + i));
145                      //cmd = teeemp;
146                      //res = teeemp;
147                     //strcpy(res, cmd);
148                     if(i == num)
149                      strcpy(res, *(temp + i));
150                     free(*(temp + i));
151                 }
152                   //printf("\n");
153                 free(temp);
154             }
155     //return cmd;
156 }
157 
158 int use_token(char* str, int num, char* res){
159     const char s[2] = "#";
160     char *token;
161     int i = 0;
162 
163     token = strtok(str, s);
164     if(num==0){
165        res = token;
166        return 0;
167     }
168     while(token != NULL){
169        //printf("i:%d, t:%s", i, token);
170        i++;
171        token = strtok(NULL, s);
172        if(num==i){
173           res = token;
174           return 0;
175        }
176     }
177     return 0;
178 }
179 
180 void get_args(char* inp_str, char* arg[]){//buffer and array
        to write
181     int i = 0;
182     const char s[2] = "#";
183     char *token;
184     token = strtok(inp_str, s);
185     arg[i] = token;
186     while(token != NULL){
```

```
187         i++;
188         token = strtok(NULL, s);
189         arg[i] = token;
190     }
191     i--;
192     arg[i] = NULL;
193 }
194
195 int main(int argc, char *argv[])
196 {
197
198     int com,p;
199
200     //printf("serv: %s port: %s", argv[1], argv[2]);
201      struct sockaddr_in si_other;
202      int s, i, slen=sizeof(si_other);
203      char buf[BUFLEN];
204      char message[BUFLEN];
205
206      if ( (s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
207      {
208          die("socket");
209      }
210
211      memset((char *) &si_other, 0, sizeof(si_other));
212      si_other.sin_family = AF_INET;
213      si_other.sin_port = htons(PORT);
214
215      if (inet_aton(argv[1] , &si_other.sin_addr) == 0)
216      {
217          fprintf(stderr, "inet_aton()␣failed\n");
218          exit(1);
219      }
220      int recv_len;
221 /*
222      int recv_len;
223      if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct
             sockaddr *) &si_other, &slen)) == -1)
224              {
225                  die("recvfrom()");
226              }
227
228              //print details of the client/peer and the data
                     received
229              //printf("Received packet from %s:%d\n",
                     inet_ntoa(si_other.sin_addr), ntohs(si_other.
                     sin_port));
230              printf("From serv: %s\n" , buf);
231
```

```
232             //now reply the client with the same data
233             if (sendto(s, buf, recv_len, 0, (struct sockaddr
                    *) &si_other, slen) == -1)
234             {
235                 die("sendto()");
236             }
237             bzero(buf, sizeof(buf));
238 */
239
240     socklen_t slen_o = sizeof(si_other);
241
242     strcpy(message, "Hello");
243
244     if (sendto(s, message, strlen(message) , 0 , (struct
            sockaddr *) &si_other, slen_o)==-1)
245                 {
246                     die("sendto()");
247                 }
248
249                 //receive a reply and print it
250                 //clear the buffer by filling null, it might
                        have previously received data
251                 memset(buf,'\0', BUFLEN);
252                 //try to receive some data, this is a
                        blocking call
253                 if (recvfrom(s, buf, BUFLEN, 0, (struct
                        sockaddr *) &si_other, &slen_o) == -1)
254                 {
255                     die("recvfrom()");
256                 }
257
258                 puts(buf);
259
260                 if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (
                        struct sockaddr *) &si_other, &slen_o)) ==
                         -1)
261                     {
262                         die("recvfrom()");
263                     }
264
265                     //print details of the client/peer
                            and the data received
266                     //printf("Received packet from %s:%d\
                            n", inet_ntoa(si_other.sin_addr),
                            ntohs(si_other.sin_port));
267                     printf("%s\n" , buf);//from server
268
269                     //now reply the client with the same
                            data
```

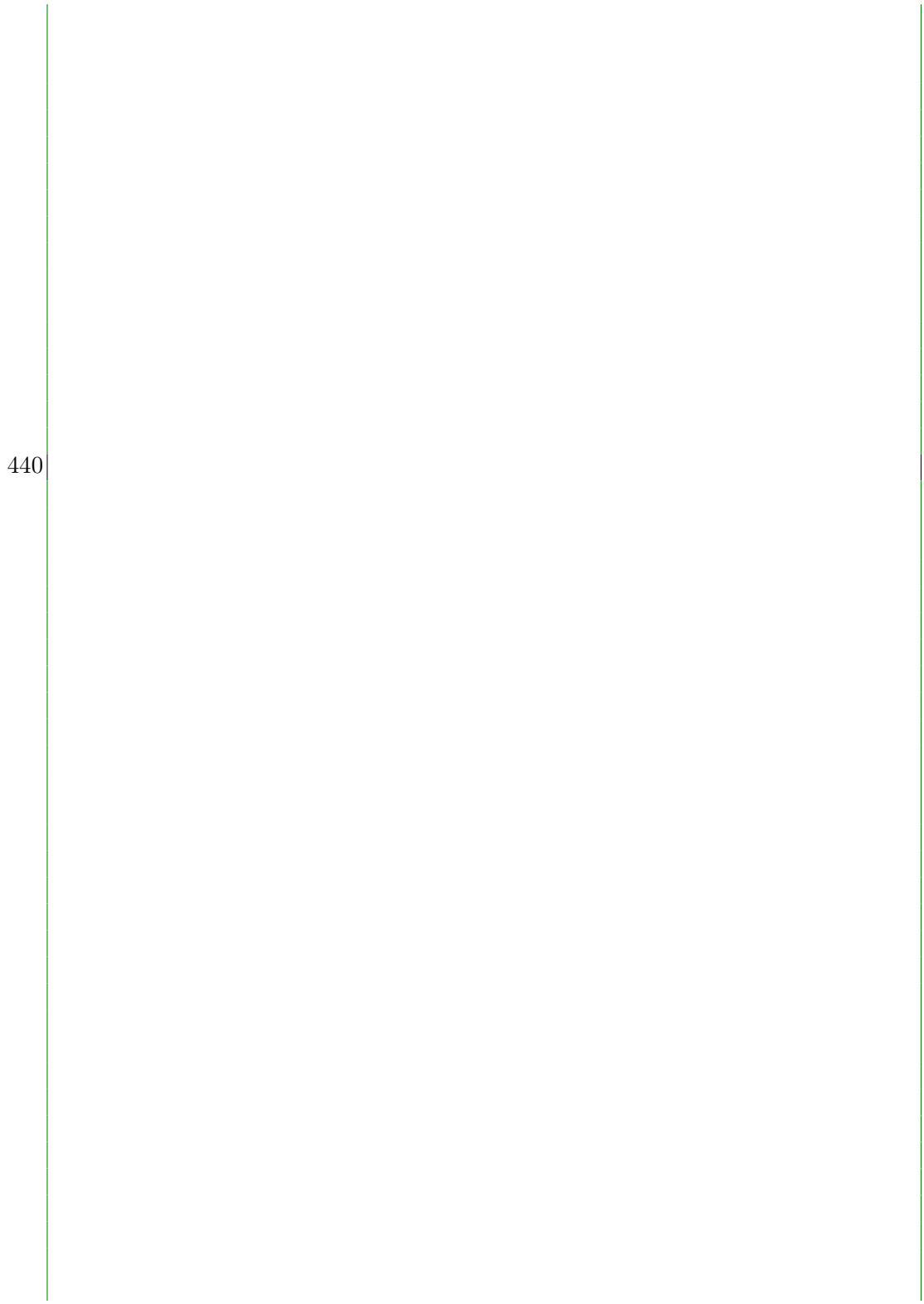
```
270                         if (sendto(s, buf, recv_len, 0, (
                                struct sockaddr*) &si_other,
                                slen_o) == -1)
271                         {
272                             die("sendto()");
273                         }
274                         bzero(buf, sizeof(buf));
275
276     bzero(message, sizeof(message));
277     //printf("Enter message : ");
278             gets(message);
279
280             //send the message
281
282             if (sendto(s, message, strlen(message) , 0 , (
                    struct sockaddr *) &si_other, slen_o)==-1)
283             {
284                 die("sendto()");
285             }
286
287             //receive a reply and print it
288             //clear the buffer by filling null, it might have
                     previously received data
289             memset(buf,'\0', BUFLEN);
290             //try to receive some data, this is a blocking
                    call
291             if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr
                    *) &si_other, &slen_o) == -1)
292             {
293                 die("recvfrom()");
294             }
295
296             //puts(buf);
297
298
299     strcpy(message, "Start");
300
301     if (sendto(s, message, strlen(message) , 0 , (struct
            sockaddr *) &si_other, slen_o)==-1)
302             {
303                 die("sendto()");
304             }
305
306             //receive a reply and print it
307             //clear the buffer by filling null, it might have
                     previously received data
308             memset(buf,'\0', BUFLEN);
309             //try to receive some data, this is a blocking
                    call
```

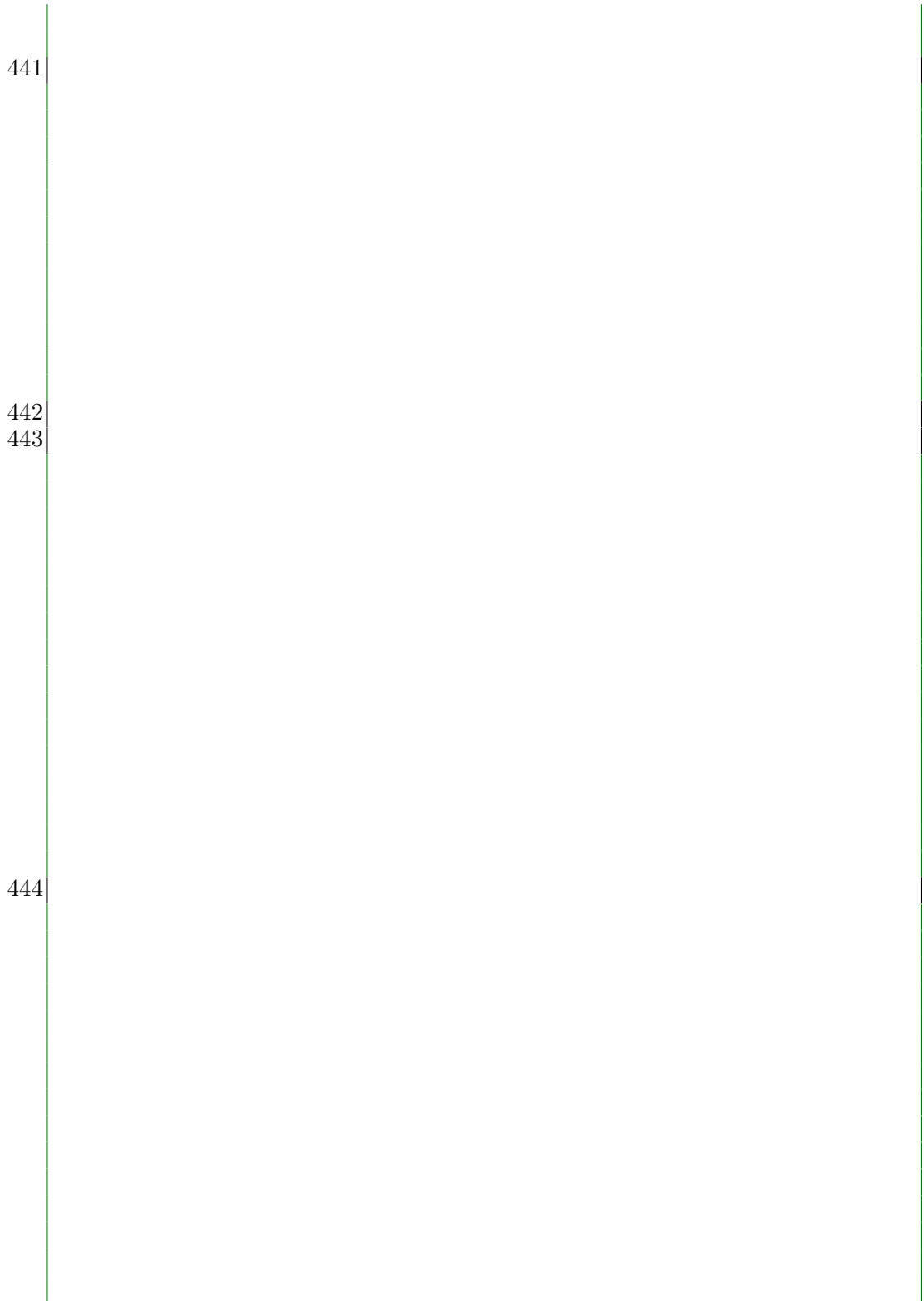
```
310             if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr
                    *) &si_other, &slen_o) == -1)
311             {
312                 die("recvfrom()");
313             }
314
315             puts(buf);
316
317             if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (
                    struct sockaddr *) &si_other, &slen_o)) == -1)
318                 {
319                     die("recvfrom()");
320                 }
321
322                 //print details of the client/peer and
                        the data received
323                 //printf("Received packet from %s:%d\n",
                        inet_ntoa(si_other.sin_addr), ntohs(
                        si_other.sin_port));
324                 printf("%s\n" , buf);
325
326                 //now reply the client with the same data
327                 if (sendto(s, buf, recv_len, 0, (struct
                        sockaddr*) &si_other, slen_o) == -1)
328                 {
329                     die("sendto()");
330                 }
331                 bzero(buf, sizeof(buf));
332                 char* arg[4];
333                 char** arg_spli;
334                 char** qq;
335                 char* arg1;
336                 char* arg2;
337
338
339     while(1)
340     {
341         //printf("Enter message : ");
342         gets(message);
343
344         //send the message
345
346         if (sendto(s, message, strlen(message) , 0 , (struct
                sockaddr *) &si_other, slen_o)==-1)
347         {
348             die("sendto()");
349         }
350
351         //receive a reply and print it
```

```
352         //clear the buffer by filling null, it might have
                previously received data
353         memset(buf,'\0', BUFLEN);
354         //try to receive some data, this is a blocking call
355         if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &
                si_other, &slen_o) == -1)
356         {
357             die("recvfrom()");
358         }
359
360         //puts(buf);
361
362 /*
363         qq=str_split(buf, '#');
364
365         if(qq){
366          int i;
367                   for (i = 0; *(qq + i); i++)
368                   {
369                       //printf("inp :%s\n", *(qq + i));
370
371                       if(i == 0)
372                           com = atoi(*(qq + i));
373                       free(*(qq + i));
374                   }
375                   //printf("\n");
376                   free(qq);
377         }*/
378
379         int spli;
380         arg_spli = splito_array(buf);
381         //for(spli=0;spli<sizeof(arg_spli);spli++)
382          //printf("aa: %s\n", arg_spli[spli]);
383         com = get_cmd(buf, 0);
384         //get_cmd_ch(buf, 1, arg1);
385         //get_cmd_ch(buf, 2, arg2);
386         //use_token(buf, 1, arg1);
387         //use_token(buf, 2, arg2);
388         //printf("test arg1: %s\n", arg1);
389         //printf("test arg2: %s\n", arg2);
390         //printf("com: %d",com);
391
392         //get_args(buf, arg);
393         //for(p = 0; p<4;p++)
394          //printf("i = %d inp: %s\n", i, arg[p]);
395         //puts(arg[0]);
396         //if(arg[0]!=NULL){
397         //   com = atoi(arg[0]);
398         //}
```

```
399         if(com == 1){
400          //puts("CL: coma 1");
401         }
402         else if(com == 2){
403          while(strcmp(buf, "allmesgs") != 0){
404          if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct
                 sockaddr *) &si_other, &slen_o)) == -1)
405                                 {
406                                     die("recvfrom()");
407                                 }
408
409                                 //print details of the client/
                                    peer and the data received
410                                 //printf("Received packet from %
                                    s:%d\n", inet_ntoa(si_other.
                                    sin_addr), ntohs(si_other.
                                    sin_port));
411                                 printf("%s\n" , buf);
412
413                                 //now reply the client with the
                                    same data
414                                 if (sendto(s, buf, recv_len, 0,
                                    (struct sockaddr*) &si_other,
                                     slen_o) == -1)
415                                 {
416                                     die("sendto()");
417                                 }
418
419          }
420                             /*if ((recv_len = recvfrom(s, buf,
                                    BUFLEN, 0, (struct sockaddr *)
                                    &si_other, &slen_o)) == -1)
421                                                 {
422                                                     die("
                                                        recvfrom
                                                        ()");
423                                                 }
424
425                                                 //print
                                                    details of
                                                    the client/
                                                    peer and
                                                    the data
                                                    received
426                                                 //printf("
                                                    Received
                                                    packet from
                                                     %s:%d\n",
                                                    inet_ntoa(
```

```
                                    si_other.
                                    sin_addr),
                                    ntohs(
                                    si_other.
                                    sin_port));
427                                 printf("%s\n"
                                    , buf);
428
429                                 //now reply
                                    the client
                                    with the
                                    same data
430                                 if (sendto(s,
                                    buf,
                                    recv_len,
                                    0, (struct
                                    sockaddr*)
                                    &si_other,
                                    slen_o) ==
                                    -1)
431                                 {
432                                     die("
                                        sendto
                                        ()");
433                                 }*/
434                               /*if ((recv_len
                                    = recvfrom(s,
                                    buf, BUFLEN,
                                    0, (struct
                                    sockaddr *) &
                                    si_other, &
                                    slen_o)) ==
                                    -1)
435
436
437
438
439
```


440


441
442
443
444

```
445
446
447
448                             //puts(buf);
449                             bzero(buf, sizeof(buf));
450
451          //puts("CL: coma 2");
452
453         }
454         else if(com == 3){
455          puts("Bye!\n");
456          exit(1);
457         }
458         else
459          puts("Wrong input\n");
460
461         /*if(strcmp(message, "Start") == 0){
462             //
463             if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (
                    struct sockaddr *) &si_other, &slen_o)) == -1)
464                 {
465                     die("recvfrom()");
466                 }
```

```
467
468                     //print details of the client/peer and
                            the data received
469                     //printf("Received packet from %s:%d\n",
                            inet_ntoa(si_other.sin_addr), ntohs(
                            si_other.sin_port));
470                     printf("From serv: %s\n" , buf);
471
472                     //now reply the client with the same data
473                     if (sendto(s, buf, recv_len, 0, (struct
                            sockaddr*) &si_other, slen_o) == -1)
474                     {
475                         die("sendto()");
476                     }
477                     bzero(buf, sizeof(buf));
478         }*/
479         bzero(message, sizeof(message));
480     }
481
482     close(s);
483     return 0;
484 }
```

5.2.8 Клиент UDP. Файл сборки Makefile

```
1 #it's my makefile
2 all:
3    #gcc -I/home/user/Downloads/libxml2-2.7.8/include main.c -
        Wall -g -O0 -L/home/user/Downloads/libxml2-2.7.8 -lxml2
         -o server1
4    #gcc -g -o0 -Wall -o ./Debug/Client1  main.c -lxml2 -I/
        home/user/Downloads/libxml2-2.7.8/include
5    gcc -g -o0 -Wall -o ./Client3  main.c -lxml2 -I/home/user/
        Downloads/libxml2-2.7.8/include
6
7 clean:
8    rm ./Client3
9    rm *.o
```

5.2.9 Сервер Windiws. Основной файл программы main.c

```
1 #define _CRT_SECURE_NO_DEPRECATE
2 #include <stdlib.h>
3 #include <stdio.h>
```

```
4  #include <sys/types.h>
5  #include <assert.h>
6  #include <winsock2.h>
7  #include <ws2tcpip.h>
8  #include <string.h>
9
10 #include <libxml\parser.h>
11 #include <libxml\tree.h>
12 #include <libxml\xmlmemory.h>
13 #include <libxml\xpath.h>
14
15
16 #pragma comment (lib, "Ws2_32.lib")
17 #pragma comment (lib, "Mswsock.lib")
18 #pragma comment (lib, "AdvApi32.lib")
19 #pragma comment (lib, "libxml2.lib")
20
21 #define MY_ENCODING "ISO-8859-1"
22 #define MY_PORT 5001
23 #define MAXUSERS 5
24
25 DWORD dwThreadId[MAXUSERS];
26
27 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
28
29 /*void add_mes(xmlNode* node, char* from, char* msg){
30    xmlNode* cur_node = NULL;
31    char buf[256];
32    bzero(buf, 256);
33    for (cur_node = node; cur_node; cur_node = cur_node->next)
34    {
35       if (cur_node->type == XML_ELEMENT_NODE) {
36          if ((!xmlStrcmp(cur_node->name, (const xmlChar *)"
              inbox"))){
37             xmlNodePtr nNode = xmlNewNode(0, (const xmlChar
                 *)"mes");
38             xmlSetProp(nNode, (const xmlChar *)"from", (const
                  xmlChar *)from);
39             xmlSetProp(nNode, (const xmlChar *)"text", (const
                  xmlChar *)msg);
40             //xmlNodeSetContent(nNode, (xmlChar*)msg);
41             xmlAddChild(cur_node, nNode);
42             return;
43          }
44       }
45       add_mes(cur_node->children, from, msg);
46    }
47 }
48
```

```
49 void create_file(const char *to, char *from, char *msg){
50    //int rc;
51    //xmlTextWriterPtr writer;
52    xmlDocPtr doc;
53    xmlNodePtr node, root;
54    //const char *UserFilename = "users.xml"
55    //xmlChar *tmp;
56    if (doc = xmlReadFile(to, NULL, 0)){
57       root = xmlDocGetRootElement(doc);
58       add_mes(root, from, msg);
59       xmlSaveFile(to, doc);
60       xmlFreeDoc(doc);
61    }
62    else{
63       doc = xmlNewDoc(BAD_CAST XML_DEFAULT_VERSION);
64       node = xmlNewDocNode(doc, NULL, BAD_CAST "inbox", NULL)
          ;
65       xmlDocSetRootElement(doc, node);
66       add_mes(node, from, msg);
67       xmlSaveFile(to, doc);
68       xmlFreeDoc(doc);
69    }
70    xmlCleanupParser();
71 }*/
72
73 /*void main(){
74    //create_file("name", "serg", "hello");
75
76 }*/
77
78 void print_mesgs(char* buffer, int newsockfd, char* user);
79 void create_file(char *to, char *from, char *msg);
80 void start_work(int newsockfd);
81
82 /*#include <stdlib.h>
83 #include <stdio.h>
84 #include <sys/types.h>
85 #include <sys/socket.h>
86 #include <netinet/in.h>
87
88 #include <libxml/parser.h>
89 #include <libxml/tree.h>
90 //#include <libxml/xmlwriter.h>
91 #include <libxml/xmlmemory.h>
92 #include <libxml/xpath.h>
93
94 #include <sys/stat.h>
95 #include <unistd.h>
96 #include <string.h>
```

```
97  #include <pthread.h>
98
99  #define MY_ENCODING "ISO-8859-1"
100 #define MY_PORT 5001
101 */
102 void get_args(char* inp_str, char* arg[]){//buffer and array
        to write
103     int i = 0, j;
104     const char s[2] = "#";
105     char *token;
106     token = strtok(inp_str, s);
107     arg[i] = token;
108     while (token != NULL){
109         i++;
110         token = strtok(NULL, s);
111         arg[i] = token;
112     }
113     i--;
114     arg[i] = NULL;
115 }
116
117 char *get_mes_from_client(int newsockfd, char *buffer){
118     int n;
119     bzero(buffer, 256);
120     n = read(newsockfd, buffer, 255);
121     if (n < 0)
122     {
123         perror("ERROR reading from socket");
124         exit(1);
125     }
126     return buffer;
127 }
128
129 char* get_hello(int newsockfd, char *buffer, char* arg[]){
130     int i, j = 0;
131     char user[10] = "no user";
132     if (arg[0] != NULL)
133         strcpy(user, arg[0]);
134     else
135         puts("No username!");
136     /*if((arg[0]!=NULL)&&(arg[1]!=NULL)){
137     strcpy(user, arg[0]);
138     char command[7];
139
140     strcpy(command, arg[1]);
141     printf("com=%s\n",arg[1]);
142
143     j = atoi(command);
144     printf("j=%d\n",j);
```

```
145     if (j==1){
146     do_send(user, arg[2], arg[3]);
147     }
148     else if (j==2){
149     do_read(newsockfd, buffer, user);
150     }
151     else if (j==3){
152     do_exit();
153     }
154     else {
155     puts("Wrong command!\n");
156     }
157     }*/
158     printf("user: %s\n", user);
159     return user;
160 }
161 
162 void do_send(char* from, char* to, char* mes){
163     printf("Send command\n");
164     create_file(to, from, mes);
165 }
166 
167 void do_read(int newsockfd, char *buffer, char* client){
168     printf("Read command\n");
169     print_mesgs(buffer, newsockfd, client);
170 }
171 
172 void do_exit(char* user){
173     printf("Exit command from user: %s\n", user);
174     //exit(1);
175 }
176 
177 void do_serve(int newsockfd, char *buffer, char* arg[], char*
        user){
178     int i, j = 0;
179     if (arg[0] != NULL){
180         char command[7];
181         strcpy(command, arg[0]);
182         j = atoi(command);
183         printf("j=%d\n", j);
184         if (j == 1){
185             do_send(user, arg[1], arg[2]);
186         }
187         else if (j == 2){
188             do_read(newsockfd, buffer, user);
189             //printf("us: %d\n", strcmp(user,"root"));
190         }
191         else if (j == 3){
192             if (strcmp(user, "root") == 0){
```

```
193                 do_exit(user);
194                 exit(1);
195             }
196             else
197                 do_exit(user);
198         }
199         else {
200             puts("Wrong command!\n");
201         }
202     }
203 }
204
205 void do_wait_mes(int newsockfd, char *buffer, char* user){
206     char* inp;
207     char* get_inp[4];
208     while (1){
209         inp = get_mes_from_client(newsockfd, buffer);
210         get_args(inp, get_inp);
211         do_serve(newsockfd, buffer, get_inp, user);
212     }
213 }
214
215 void add_mes(xmlNode* node, char* from, char* msg){
216     xmlNode* cur_node = NULL;
217     char buf[256];
218     bzero(buf, 256);
219     for (cur_node = node; cur_node; cur_node = cur_node->next)
220     {
221         if (cur_node->type == XML_ELEMENT_NODE) {
222             if ((!xmlStrcmp(cur_node->name, (const xmlChar *)"
                    inbox"))){
223                 xmlNodePtr nNode = xmlNewNode(0, (const xmlChar
                        *)"mes");
224                 xmlSetProp(nNode, (const xmlChar *)"from", (const
                        xmlChar *)from);
225                 xmlSetProp(nNode, (const xmlChar *)"text", (const
                        xmlChar *)msg);
226                 //xmlNodeSetContent(nNode, (xmlChar*)msg);
227                 xmlAddChild(cur_node, nNode);
228                 return;
229             }
230         }
231         add_mes(cur_node->children, from, msg);
232     }
233 }
234
235 void read_mes(xmlNode* node, char* mess, int newsockfd){
236     xmlNode *cur_node = NULL;
237     char buf[256];
```

```
238     bzero(buf, 256);
239     int n;
240     for (cur_node = node; cur_node; cur_node = cur_node->next)
            {
241         if (cur_node->type == XML_ELEMENT_NODE) {
242             if ((!xmlStrcmp(cur_node->name, (const xmlChar *)"
                    mes")))
243             {
244                 strcpy(buf, "from: ");
245                 strncat(buf, xmlGetProp(cur_node, "from"), strlen
                        (xmlGetProp(cur_node, "from")));
246                 strcat(buf, "  message: ");
247                 strncat(buf, xmlGetProp(cur_node, "text"), strlen
                        (xmlGetProp(cur_node, "text")));
248                 strcat(buf, "\n");
249                 n = write(newsockfd, buf, strlen(buf));
250                 //n=read(newsockfd,buf,255);
251             }
252         }
253
254         read_mes(cur_node->children, mess, newsockfd);
255     }
256 }
257
258 void print_mesgs(char* buffer, int newsockfd, char* user)
259 {
260
261     xmlDoc         *doc = NULL;
262     xmlNode        *root_element = NULL;
263     int n;
264     doc = xmlReadFile(user, NULL, 0);
265     if (doc == NULL)
266     {
267         printf("error: could not parse file %s\n", user);
268         //strncat(buffer,"Error\n",6);
269         n = write(newsockfd, "Error\n", 6);
270     }
271     else
272     {
273
274         root_element = xmlDocGetRootElement(doc);
275         n = write(newsockfd, "Messages:\n", 11);
276         read_mes(root_element, buffer, newsockfd);
277         //n = write(newsockfd,"end",3);
278         xmlFreeDoc(doc);
279     }
280
281     xmlCleanupParser();
282
```

```
283     //return;
284
285 }
286
287 void create_file(char *to, char *from, char *msg){
288     int rc;
289     //xmlTextWriterPtr writer;
290     xmlDocPtr doc;
291     xmlNodePtr node, root;
292     xmlChar *tmp;
293     if (doc = xmlReadFile(to, NULL, 0)){
294         root = xmlDocGetRootElement(doc);
295         add_mes(root, from, msg);
296         xmlSaveFile(to, doc);
297         xmlFreeDoc(doc);
298     }
299     else{
300         doc = xmlNewDoc(BAD_CAST XML_DEFAULT_VERSION);
301         node = xmlNewDocNode(doc, NULL, BAD_CAST "inbox", NULL)
                ;
302         xmlDocSetRootElement(doc, node);
303         add_mes(node, from, msg);
304         xmlSaveFile(to, doc);
305         xmlFreeDoc(doc);
306     }
307     xmlCleanupParser();
308 }
309 struct sockParams
310 {
311     int sockfd, newsockfd, port_number, client;
312     struct sockaddr_in serv_addr, cli_addr;
313 };
314
315 struct userThread
316 {
317     char login[MAXUSERS][100];
318     int count;
319 };
320
321 DWORD WINAPI startThread(LPVOID in)
322 {
323     struct sockParams *sp = (struct sockParams *)in;
324     start_work(sp->newsockfd);
325     return 0;
326 }
327
328 void start_work(int newsockfd){
329     char buffer[256];
330     int n;
```

```
331     n = write(newsockfd, "Please, write ur name ending with
            '#'\n", 38);
332
333     //bzero(buffer,256);
334     bzero(buffer, 256);
335     n = read(newsockfd, buffer, 255);
336     if (n < 0)
337     {
338        perror("ERROR reading from socket");
339        exit(1);
340     }
341     //int j;
342     char* get_inp[4];
343     get_args(buffer, get_inp);
344     char user_name[10] = "nnm";
345     strcpy(user_name, get_hello(newsockfd, buffer, get_inp));
346     n = write(newsockfd, "Commands: 1#kitty#Hello# - Send
            message 'Hello' to user kitty;\n"
347        "2# - read ur inbox; 3# - exit\n", 93);
348     do_wait_mes(newsockfd, buffer, user_name);
349     //return 0;
350 }
351
352 struct userThread usersthr;
353
354 DWORD WINAPI workMainTh(LPVOID lpParam)
355 {
356     char buffer[10];
357     int i;
358     while (1)
359     {
360        fgets(buffer, 10 - 1, stdin);
361        for (i = 0; i<MAXUSERS; i++)
362        {
363           if (strcmp(buffer, usersthr.login[i]))
364              ExitThread(dwThreadId[i]);
365        }
366     }
367     return 0;
368 }
369
370 int main(int argc, char *argv[])
371 {
372     HANDLE thread[MAXUSERS], mainthread;
373     //pthread_t thread[5], mainthread;
374     int i = 0, j = 0;
375     struct sockParams sp;
376
377     WSADATA wsaData;
```

```
378     WSAStartup(MAKEWORD(2, 2), &wsaData);
379     usersthr.count = 0;
380
381     sp.sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
382     if (sp.sockfd < 0)
383     {
384        perror("ERROR opening socket");
385        exit(1);
386     }
387     bzero((char *)&sp.serv_addr, sizeof(sp.serv_addr));
388     sp.port_number = MY_PORT;
389     //portno = 7771;
390     sp.serv_addr.sin_family = AF_INET;
391     sp.serv_addr.sin_addr.s_addr = INADDR_ANY;
392     sp.serv_addr.sin_port = htons(sp.port_number);
393
394     /* Now bind the host address using bind() call.*/
395     if (bind(sp.sockfd, (struct sockaddr *) &sp.serv_addr,
396        sizeof(sp.serv_addr)) < 0)
397     {
398        perror("ERROR on binding");
399        exit(1);
400     }
401
402     /* Now start listening for the clients, here process will
403     * go in sleep mode and will wait for the incoming
           connection
404     */
405     while (1){
406        listen(sp.sockfd, 5);
407        sp.client = sizeof(sp.cli_addr);
408
409        //printf("cl: %s\n", sp.client);
410        /* Accept actual connection from the client */
411        sp.newsockfd = accept(sp.sockfd, (struct sockaddr *)&sp
              .cli_addr, &sp.client);
412
413        sp.port_number = i;
414
415        /*if (sp.newsockfd < 0)
416        {
417        perror("ERROR on accept");
418        exit(1);
419        }
420        pthread_create(&thread[i], NULL, startThread, (void*)&
              sp);
421        i++;
422        }
423
```

```
424         for (j = 0; j<5; j++)
425         pthread_join(thread[j], NULL);*/
426
427         thread[i] = CreateThread(NULL, 0, startThread, (LPVOID)
                &sp, 0, &dwThreadId[i]);
428         i++;
429     }
430     return 0;
431 }
```

5.2.10 Клиент Windiws. Основной файл программы main.c

```
1  //#include <stdio.h>
2  //#include <sys/types.h>
3  //#include <sys/socket.h>
4  //#include <netinet/in.h>
5
6  #define _CRT_SECURE_NO_WARNINGS
7  #define _WINSOCK_DEPRECATED_NO_WARNINGS
8  #include <time.h>
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <winsock2.h>
13 #include <ws2tcpip.h>
14 #include <string.h>
15 #include <assert.h>
16
17 #pragma comment (lib, "Ws2_32.lib")
18 #pragma comment (lib, "Mswsock.lib")
19 #pragma comment (lib, "AdvApi32.lib")
20
21 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
22 #define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void)
       0)
23
24 void print_mesgs(char* buffer, int sockfd, char* user);
25
26 void write_socket_start(char* buffer, int sockfd){
27     int n;
28     bzero(buffer, 256);
29     fgets(buffer, 255, stdin);
30     //puts(buffer);
31     /* Send message to the server */
32     n = write(sockfd, buffer, strlen(buffer));
33     if (n < 0)
```

```
34      {
35          perror("ERROR writing to socket");
36          exit(1);
37      }
38  }
39
40  void read_response(char* buffer, int sockfd){
41      int n;
42      bzero(buffer, 256);
43      n = read(sockfd, buffer, 255);
44      if (n < 0)
45      {
46          perror("ERROR reading from socket");
47          exit(1);
48      }
49      printf("%s", buffer);
50  }
51  //-----------------------------------------
52  void get_args(char* inp_str, char* arg[]){//buffer and array
        to write
53      int i = 0, j;
54      const char s[2] = "#";
55      char *token;
56      token = strtok(inp_str, s);
57      arg[i] = token;
58      while (token != NULL){
59          i++;
60          token = strtok(NULL, s);
61          arg[i] = token;
62      }
63      i--;
64      arg[i] = NULL;
65  }
66
67  char *get_mes_from_client(int newsockfd, char *buffer){
68      int n;
69      bzero(buffer, 256);
70      n = read(newsockfd, buffer, 255);
71      if (n < 0)
72      {
73          perror("ERROR reading from socket");
74          exit(1);
75      }
76      return buffer;
77  }
78
79  void resp_send(char* from, char* to, char* mes){
80  }
81
```

```
82 void resp_read(int newsockfd, char *buffer, char* client){
83    print_mesgs(buffer, newsockfd, client);
84 }
85
86 void resp_exit(){
87    exit(1);
88 }
89
90 void print_mesgs(char* buffer, int sockfd, char* user){
91    read_response(buffer, sockfd);
92    read_response(buffer, sockfd);
93 }
94
95 char* put_hello(int sockfd, char *buffer){
96    write_socket_start(buffer, sockfd);
97    char* arg[4];
98    get_args(buffer, arg);
99    int i, j = 0;
100   char user[10] = "no user";
101   if ((arg[0] != NULL) && (arg[1] != NULL)){
102      strcpy(user, arg[0]);
103      char command[7];
104      strcpy(command, arg[1]);
105      j = atoi(command);
106      if (j == 1){
107         resp_send(user, arg[2], arg[3]);
108      }
109      else if (j == 2){
110         resp_read(sockfd, buffer, user);
111      }
112      else if (j == 3){
113         resp_exit();
114      }
115      else {
116         puts("Wrong command!\n");
117      }
118   }
119   return user;
120 }
121
122 void keep_talking(int sockfd, char *buffer){
123   write_socket_start(buffer, sockfd);
124   char* arg[4];
125   get_args(buffer, arg);
126   int i, j = 0;
127   char user[10] = "no user";
128   if ((arg[0] != NULL)){
129      strcpy(user, arg[0]);
130      char command[7];
```

```
131         strcpy(command, arg[0]);
132         j = atoi(command);
133         if (j == 1){
134             resp_send(user, arg[2], arg[3]);
135         }
136         else if (j == 2){
137             resp_read(sockfd, buffer, user);
138         }
139         else if (j == 3){
140             resp_exit();
141         }
142         else {
143             puts("Wrong command!\n");
144         }
145     }
146 }
147 
148 int main(int argc, char *argv[])
149 {
150     SOCKET sockfd;
151     int portno, n;
152     struct sockaddr_in serv_addr;
153     struct hostent *server;
154 
155     char buffer[256];
156 
157     WSADATA wsaData;
158 
159     WSAStartup(MAKEWORD(2, 2), &wsaData);
160 
161     if (argc < 3) {
162         fprintf(stderr, "usage %s hostname port\n", argv[0]);
163         exit(0);
164     }
165     portno = atoi(argv[2]);
166     /* Create a socket point */
167     sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
168     if (sockfd == INVALID_SOCKET)
169     {
170         perror("ERROR opening socket");
171         exit(1);
172     }
173     server = gethostbyname(argv[1]);
174     if (server == NULL) {
175         fprintf(stderr, "ERROR, no such host\n");
176         exit(0);
177     }
178 
179     bzero((char *)&serv_addr, sizeof(serv_addr));
```

```
180     serv_addr.sin_family = AF_INET;
181
182     bcopy((char *)server->h_addr,
183     (char *)&serv_addr.sin_addr.s_addr,
184     server->h_length);
185
186     serv_addr.sin_port = htons(portno);
187
188     /* Now connect to the server */
189     //if (connect(sockfd, (sockaddr *)&serv_addr, sizeof(
            serv_addr))>0)
190
191     if(connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(
            serv_addr)) > 0)
192     {
193         perror("ERROR connecting");
194         exit(1);
195     }
196     /* Now ask for a message from the user, this message
197     * will be read by server
198     */
199     read_response(buffer, sockfd);
200     put_hello(sockfd, buffer);
201     read_response(buffer, sockfd);
202     while (1)
203         keep_talking(sockfd, buffer);
204     return 0;
205 }
```