Сети ЭВМ и телекоммуникации

А. И. Баратынский

26 декабря 2014 г.

Глава 1

Задание

Разработать приложение-клиент и приложение сервер, обеспечивающие функции обмена файлами.

1.1 Функциональные требования

Серверное приложение должно реализовывать следующие функции:

1. Прием файла от клиента
2. Передача по запросу от клиента списка файлов текущего каталога
3. Прием запросов на передачу файла и передача файла клиенту
4. Навигация по системе каталогов
5. Обработка запроса на отключение клиента
6. Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1. Получение от сервера списка файлов каталога
2. Операции навигации по системе каталога
3. Передача файла серверу
4. Прием файла от сервера
5. Разрыв соединения
6. Обработка ситуации отключения клиента сервером

1.2 Нефункциональные требования

Серверное приложение:

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

Клиентское приложение должно реализовывать следующие функции:

1. Установление соединения с сервером

1.3 Накладываемые ограничения

1. Размер передаваемого файла не должен превышать 1 Гб
2. Имя передаваемого файла должно быть не более 30 символов

Глава 2

Реализация для работы по протоколу TCP

2.1 Прикладной протокол

Первое сообщение клиента всегда имя пользователя. После этого на клиентской стороне создается каталог данного пользователя (в случае, если каталог уже существует, то клиентское приложение просто переходит в него), и начинается дальнейшее взаимодействие. Далее, с помощью функции getFirstMessage() сервер определяет, какое сообщение было отправлено клиентом.

ls При таком сообщении от клиента, сервер посылает список всех доступных файлов в данном каталоге. Клиент определяет, что весь список файлов получен, когда приходит сообщение с символом &.

get При таком сообщении от клиента, сервер ожидает нового сообщения с именем передаваемого файла. Когда такое сообщение получено, происходит передача файла. Клиент определяет, что передача файла закончена, когда приходит сообщение с символом &.

transfer Это передача файла от клиента серверу. Первым передается имя файла, затем его размер. После этого начинается передача.

mkdir Создание директории на сервере. Вторым сообщением передается название директории.

cd Переход в директорию. Во втором сообщении указывается название директории.

2.2 Архитектура приложения

На сервере все файлы хранятся в специальном каталоге SERVER. На клиентской стороне для каждого пользователя создается собственный каталог, в рамках которого он и работает.
Приложение поддерживает многопоточность. На каждого клиента запускается свой поток. Серверное приложение всегда ждет какого-нибудь сообщения от клиента. После получения сообщения функция getFirstMessage() определяет, какое сообщение было получено. В зависимости от полученного сообщения, сервер вызывает его обработчик.
Клиентское приложение находится в режиме ожидания ввода очередного сообщения. Если введенное сообщение некорректно (не входит в список известных серверу команд), то клиентское приложение просто игнорирует его.
Несмотря на то, что в протоколе TCP на каждое принятое сервером сообщение по умолчанию посылается подтверждение принятого пакета, мною были добавлены свои ответы сервера. Это сказалось на производительности приложения, т.к. передаваемых пакетов стало в два раза больше, но зато обеспечило легкий переход от TCP к UDP.

Формат команд
В разработанных приложениях один принцип взаимодействия: сначала от клиента к серверу приходит сообщение с командой, а дальше передаются аргументы команды. Возможно, было бы лучше сделать передачу команды вместе с аргументами в одном пакете, т.к. плюсов в выбранном подходе нет, но такая идея пришла уже после того, как все приложение было написано, поэтому было принято решение оставить все как есть.
transfer
Сначала посылается команда transfer, затем посылается имя файла, далее посылается размер файла, далее передается тело файла пакетами определенной длины (в данном случае, по 1024 байта). Опять же, передача файлов пакетами определенной длины по протоколу TCP возможно и не самое лучшее решение, т.к. в протоколе TCP передается поток данных, но это было сделано для легкого перехода на протокол UDP, где существуют ограничения на длину пакета. На каждое присланное сообщение сервер отвечает.
get
Все происходит аналогично команде transfer, только в данном случае сообщения идут от сервера клиенту.
cd
Первым сообщением отсылается сама команда, затем директория, в ко-

торую нам нужно перейти.
ls
Клиент посылает сообщение ls серверу. Сервер обрабатывает сообщение, затем, с помощью системного вызова system("ls>ls"), все файлы текущего каталога записываются в файл ls. Далее сервер считывает из этого файла по одному имени и отправляет его клиенту. После завершения пердачи файл ls удаляется.
mkdir
Клиент посылает сообщение mkdir. Вторым сообщением он отправляет название новой директории.

2.3 Тестирование

2.3.1 Тестовый план и результаты тестирования

Взаимодействие серверного и клиентского приложений проверялось на одном компьютере. В одном терминале запускался сервер, а в других - клиенты.

1. Передача корректного файла серверу Передача существующего файла размером меньше 1 Гб завершается без ошибок.

2. Передача несуществующего файла серверу При попытке передать несуществующий файл серверу выведется сообщение об ошибке: "There is no such file!!!". После этого продолжается обычная работа приложения.

3. Передача файла размером больше 1 Гб При попытке передать файл размером больше 1 Гб серверу выведется сообщение об ошибке: "This file is too big!". После этого продолжается обычная работа приложения.

4. Передача файла с помощью команды get При вводе команды get, а затем имени нужного файла, происходит его передача от сервера к клиенту.

5. Параллельная передача файлов от двух клиентов Для данного теста в каталоги двух пользователей были помещены файлы размером по 350 мб. После выполнения команд transfer в клиентских приложених выполняется их передача на сервер в каталог SERVER.

6. Параллельная передача файлов от двух клиентов с вводом в одну из команд функции sleep(10000000000) Добавим функцию sleep(10000000000)

в обработчик команды transfer. На одном из клиентов запустим эту команду. В это время, несмотря на то, что первый клиент завис, во втором можно полноценно работать.

7. Ввод команды ls После ввода команды ls на клиентску сторону приходит список всех доступных файлов.

8. Ввод команды cd После ввода команды cd и ввода имени каталога происходит переход в это каталог. Тестирование показало, что командой cd можно выйти за переделы каталога SERVER, вплоть до корневой директории. Это несомненный минус данного приложения.

9. Ввод команды mkdir После ввода команды mkdir и ввода имени нового каталога, он создается.

10. Подключение нескольких клиентов Подключение нескольких клиентов проходит успешно. В каждом из запущенных клиентов были введены команды transfer, get и ls, на эти команды были получены корректные ответы.

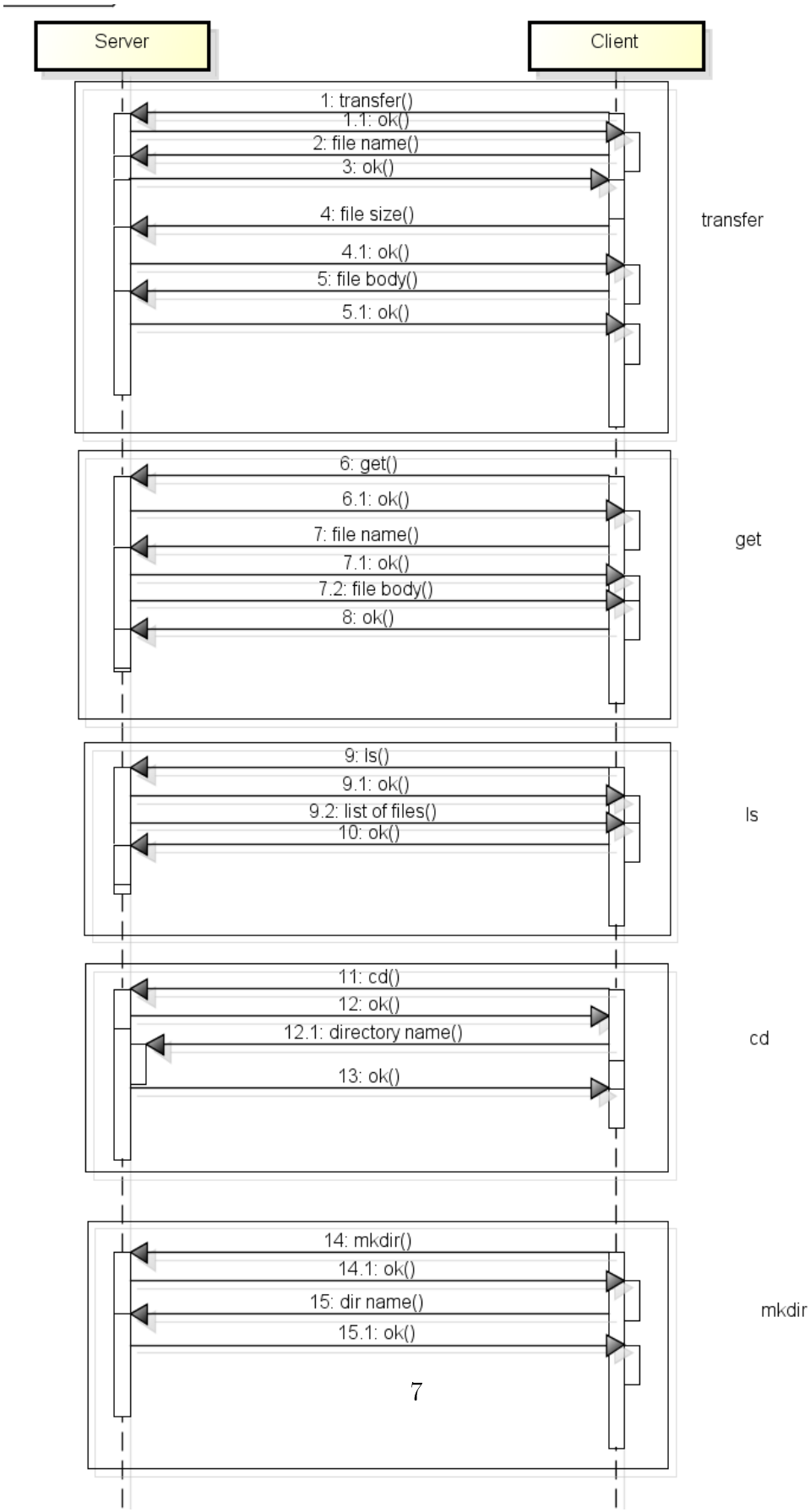

Рис. 2.1: Sequence diagram

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

Реализация для работы по протоколу UDP не притерпела особых изменений, т.к. изначально, реализация для работы по протоколу TCP была сделана для легкого перехода на протокол UDP. Также не была реализована многопоточность.
Проблема состоит в том, что при использовании UDP трудно получить IP адрес отправителя. Но было найдено другое решение, позволяющее различать клиентов между собой: с помощью функций recvfrom и sendto. Аргумент `(struct␣sockaddr*)␣&s.si_other` - это и есть адрес клиента. Таким образом, сервер может отвечать на запросы конкретному клиенту. Добавим в обработчик запросов строку:

```
printf("Received␣packet␣from␣%s:%d\n",
inet_ntoa(s.si_other.sin_addr),ntohs(s.si_other.sin_port));
```

Теперь сервер будет выводить с какого IP-адреса и порта пришел запрос в следующем формате:
Received packet from 127.0.0.1:47924
Received packet from 127.0.0.1:50207

Таким образом, мы сохраним адрес отправителя и сможем обмениваться с ним сообщениями.
Также для UDP реализована функция подсчета дошедших пакетов. На серверной стороне инициализирована переменная count, которая считает номер пакета, который должен дойти в данной итерации. На клиентской стороне в начале каждого отправляемого пакета файла отведено по три

байта на номер посылки. Сервер обрабатывает данное значение и сравнивает его со счетчиком count. Если значения не совпадают, то клиент высылает сообщение об ошибке, и сервер заново посылает нужный пакет. Первое сообщение клиента всегда имя пользователя. После этого на клиентской стороне создается каталог данного пользователя (в случае, если каталог уже существует, то клиентское приложение просто переходит в него), и начинается дальнейшее взаимодействие. Далее, с помощью функции getFirstMessage() сервер определяет, какое сообщение было отправлено клиентом.

1. ls При таком сообщении от клиента, сервер посылает список всех доступных файлов в данном каталоге. Клиент определяет, что весь список файлов получен, когда приходит сообщение с символом &.

2. get При таком сообщении от клиента, сервер ожидает нового сообщения с именем передаваемого файла. Когда такое сообщение получено, происходит передача файла. Клиент определяет, что передача файла закончена, когда приходит сообщение с символом &.

3. transfer Это передача файла от клиента серверу. Первым передается имя файла, затем его размер. После этого начинается передача. В каждой посылке первые три байта - номер отправляемого пакета. Серверная сторона обрабатывает этот номер и сравнивает с собственным счетчиком. Если значение совпадает, то сервер посылает ответ клиенту ALRIGHT, если нет - то сервер посылает сообщение ERROR. Это означает, что нужно снова отправить предыдущий пакет.

4. mkdir Создание директории на сервере. Вторым сообщением передается название директории.

5. cd Переход в директорию. Во втором сообщении указывается название директории.

Объяснение выбранной длины пакета
Теоретически, максимальный размер IP-дейтаграммы составляет 65535 байтов, что обусловлено 16-разрядным полем полной длины в IP-заголовке. При размере IP-заголовка 20 байтов и UDP-заголовка - 8 байтов остается максимум 65507 байтов для данных UDP-пакета. Однако в большинстве реализаций устанавливается более жесткое ограничение.
Любой хост обязан принимать IP-дейтаграммы длиной по крайней мере 576 байтов. Многие приложения, использующие UDP, спроектированы так, чтобы оперировать данными размером не более 512 байтов, с целью

оставаться в гарантированных пределах.
В моем приложении, я увеличил минимальный порог в два раза, чтобы немного сократить количество пакетов.

3.2 Тестирование

3.2.1 Тестовый план и результаты тестирования

Взаимодействие серверного и клиентского приложений проверялось на одном компьютере. В одном терминале запускался сервер, а в других - клиенты.

1. Передача корректного файла серверу Передача существующего файла размером меньше 1 Гб завершается без ошибок.

2. Передача несуществующего файла серверу При попытке передать несуществующий файл серверу выведется сообщение об ошибке: "There is no such file!!!". После этого продолжается обычная работа приложения.

3. Передача файла размером больше 1 Гб При попытке передать файл размером больше 1 Гб серверу выведется сообщение об ошибке: "This file is too big!". После этого продолжается обычная работа приложения.

4. Передача файла с помощью команды get При вводе команды get, а затем имени нужного файла, происходит его передача от сервера к клиенту.

5. Ввод команды ls После ввода команды ls на клиентску сторону приходит список всех доступных файлов.

6. Ввод команды cd После ввода команды cd и ввода имени каталога происходит переход в это каталог. Тестирование показало, что командой cd можно выйти за переделы каталога SERVER, вплоть до корневой директории. Это несомненный минус данного приложения.

7. Ввод команды mkdir После ввода команды mkdir и ввода имени нового каталога, он создается.

8. Подключение нескольких клиентов Подключение нескольких клиентов проходит успешно. В каждом из запущенных клиентов были

введены команды transfer, get и ls, на эти команды были получены корректные ответы.

Глава 4

Выводы

4.1 TCP

Протокол TCP является сквозным и ориентирован на создание соединений, то есть в данном протоколе создаётся виртуальное соединение. Протокол TCP размещается над сетевым протоколом IP, который даёт возможность TCP посылать и принимать сегменты информации различной длины, вложенные в межсетевые дейтаграммные «конверты» (пакеты).

При организации связи между парой прикладных процессов протокол TCP обеспечивает следующее: надёжную передачу данных; управление потоком данных; мультиплексирование; организацию, поддержание и сброс виртуального соединения (виртуального канала); приоритетную доставку информации и её безопасность.

Протокол TCP, в отличие от протокола UDP, создаёт виртуальные соединения или каналы. Подобно модулю UDP, прикладные процессы взаимодействуют с модулем TCP через порты, которые имеют общеизвестные адреса (номера).

Когда прикладной процесс начинает использовать TCP, то этот модуль на хосте и модуль TCP на сервере приложений начинают взаимодействовать. Эти два оконечных модуля, прежде всего, создают виртуальное соединение, которое является дуплексным и расходует ресурсы обоих оконечных модулей TCP. Протокол TCP разбивает поток двоичных разрядов (поступающих с вышележащего уровня) на TCP-сегменты, которые передаются по виртуальному соединению. На приёмном конце производится обратная операция.

Протокол TCP требует, чтобы все отправленные сегменты данных были подтверждены с приёмного конца, т.е. используется алгоритм об-

ратной связи. Для повышения эффективности работы используются механизм скользящего окна, тайм-ауты и повторные передачи для обеспечения надёжной доставки.

4.2 UDP

Протокол UDP называют протоколом ненадёжной доставки. Этот протокол предоставляет прикладным процессам транспортные услуги, которые немногим отличаются от услуг протокола IP (сетевого уровня).

Протокол UDP обеспечивает только доставку дейтаграммы и не гарантирует её выполнение. При обнаружении ошибки дейтаграмма просто стирается. Протокол не поддерживает виртуального соединения с удалённым модулем UDP. Чаще всего базируется на принципах динамической маршрутизации (каждая дейтаграмма передаётся по оптимальному маршруту). Основное достоинство — простота.

формат протокола UDP размещается в поле данных IP-пакета (или после заголовка IP-пакета) и содержит следующие поля:

Поле «Порт источника» (Source Port) указывает порт процесса источника, куда может быть адресован ответ на данное сообщение.

Поле «Порт получателя» (Destination Port) идентифицирует принимающий процесс. Под «портом» понимается адрес (номер) некой точки доступа к услугам другого уровня. В случае архитектуры TCP/IP под портом понимается некий номер области памяти, где размещаются передаваемые в сеть (протоколу UDP или TCP) и принимаемые из сети (поступающие в распоряжение операционной системы) данные. Номера портов на передачу и приём в общем случае могут различаться. На приёмной и передающей сторонах взаимодействие процессов в общем случае может происходить через разные номера портов, поэтому указание порта в заголовке UDP-дейтаграммы необходимо.

В поле «Длина» (Length) указывается размер данной дейтаграммы с учётом длины заголовка в байтах.

Поле «Контрольная сумма» (Checksum) обеспечивает контроль правильности данных и заголовка. Суммируются все контролируемые 16-битные слова (с циклическим переносом из старшего разряда в младший). Инвертированное значение результата записывается в поле контрольной суммы. Если UDP-дейтаграмма содержит нечетное число байтов, то недостающий последний байт в таких случаях считается нулевым. Этот байт не передается в области данных. При подсчете контрольной суммы протокол UDP учитывает 12-байтовый псевдозаголовок (pseudo header). Псевдозаголовок включает в себя: IP-адрес источника, IP-адрес

приемника, протокол (код 17) и длину UDP-дейтаграммы. Если источник проставил контрольную сумму, а адресат при ее проверке обнаружил ошибку, то UDP-дейтаграмма "молчаливо отбрасывается не генерируется никакого сообщения об ошибке.
Реализованные приложения работают хорошо, но приложение с использованием протокола UDP работает быстрее. Причины перечислены выше.

Приложения

Описание среды разработки

Linux debian 3.2.0-4-486 1 Debian 3.2.60-1+deb7u3 i686 GNU/Linux.
Среда разработки - Eclipse.
Windows 8.
Среда разработки - Visual Studio 2010.

Листинги

Linux TCP server

```
1  /*
2
3   * Server.c
4
5   *
6
7   *  Created on: Oct 16, 2014
8
9   *      Author: Baratynskiy
10
11  */
12
13
14
15 #include <stdio.h>
16
17 #include <sys/types.h>
18
19 #include <sys/socket.h>
20
21 #include <sys/stat.h>
22
23 #include <netinet/in.h>
```

```
24
25 #include <unistd.h>
26
27 #include <pthread.h>
28
29
30
31 #define bufSize 1024
32
33
34
35 struct SockParams{
36
37     int sock, bufsocket, port, clilen;
38
39     struct sockaddr_in serverAddr, clientAddr;
40
41 };
42
43
44
45 int doprocessing(int sock, FILE *f) {
46
47     char buffer[bufSize];
48
49     bzero(buffer, bufSize);
50
51     read(sock, buffer, bufSize - 1);
52
53     write(sock, "I␣got␣your␣message", 18);
54
55     fprintf(f, "%s", buffer);
56
57     return 0;
58
59 }
60
61
62
63 char* getFirstMessage(int sock, char *firstMessage) {
64
65     bzero(firstMessage, bufSize);
66
67     read(sock, firstMessage, bufSize - 1);
68
69     if (strcmp(firstMessage, "ls") == 0) {
70
71         write(sock, "ls␣command", 10);
72
```

```
73          firstMessage = "ls";
74
75          return "ls";
76
77      } else if (strcmp(firstMessage, "get") == 0) {
78
79          write(sock, "get␣command", 11);
80
81          firstMessage = "get";
82
83          return "get";
84
85      } else if (strcmp(firstMessage, "mkdir") == 0) {
86
87          write(sock, "mkdir␣command", 11);
88
89          firstMessage = "mkdir";
90
91          return "mkdir";
92
93      } else if (strcmp(firstMessage, "cd") == 0) {
94
95          write(sock, "cd␣command", 11);
96
97          firstMessage = "cd";
98
99          return "cd";
100
101     } else {
102
103         write(sock, "I␣got␣file␣name", 15);
104
105     }
106
107     return firstMessage;
108
109 }
110
111
112
113 int getFileSize(int sock, int size) {
114
115     char buffer[bufSize];
116
117     bzero(buffer, bufSize);
118
119     read(sock, buffer, bufSize - 1);
120
121     write(sock, "I␣got␣file␣size", 15);
```

```
122
123     sscanf(buffer, "%d", &size);
124
125     return size;
126
127 }
128
129
130
131 void listenToClients(int sock, int bufsocket, int clilen,
132
133         struct sockaddr_in clientAddr, char *name, int fs) {
134
135     getFirstMessage(bufsocket, name);
136
137     if (strcmp(name, "ls") == 0) {
138
139         system("ls>ls");
140
141         FILE *ls = fopen("ls", "r");
142
143         char charSize[30];
144
145         char lsStr[30], buffer[30];
146
147         while (!feof(ls)) {
148
149             fgets(lsStr, sizeof(lsStr), ls);
150
151             if (strcmp(lsStr, "ls\n") == 0 ) {
152
153                 continue;
154
155             }
156
157             write(bufsocket, lsStr, sizeof(lsStr));
158
159             read(bufsocket, buffer, 1);
160
161             bzero(lsStr, sizeof(lsStr));
162
163         }
164
165         write(bufsocket, "&", 2);
166
167         fclose(ls);
168
169         system("rm␣ls");
170
```

```
171         return;
172 
173     }
174 
175     if (strcmp(name, "get") == 0) {
176 
177         char filename[30];
178 
179         char buffer[1024];
180 
181         read(bufsocket, filename, sizeof(filename));
182 
183         write(bufsocket, filename, sizeof(filename));
184 
185         printf("%s␣is␣wanted\n", filename);
186 
187         FILE *file = fopen(filename, "r+");
188 
189         while (!feof(file)) {
190 
191             fread(buffer, 1, sizeof(buffer), file);
192 
193             write(bufsocket, buffer, sizeof(buffer));
194 
195             read(bufsocket, buffer, sizeof(buffer));
196 
197         }
198 
199         write(bufsocket, "&", 1);
200 
201         read(bufsocket, buffer, sizeof(buffer));
202 
203         fclose(file);
204 
205         return;
206 
207     }
208 
209     if (strcmp(name, "mkdir") == 0) {
210 
211         char dirname[30];
212 
213         read(bufsocket, dirname, sizeof(dirname));
214 
215         write(bufsocket, dirname, sizeof(dirname));
216 
217         printf("Directory␣%s␣was␣created\n", dirname);
218 
219         mkdir(dirname, S_IRWXU);
```

```
220
221         return;
222
223     }
224
225     if (strcmp(name, "cd") == 0) {
226
227         char dirname[30];
228
229         read(bufsocket, dirname, sizeof(dirname));
230
231         write(bufsocket, dirname, sizeof(dirname));
232
233         chdir(dirname);
234
235         return;
236
237     }
238
239     char rmfile[70];
240
241     bzero(rmfile,sizeof(rmfile));
242
243     sscanf(name, "rm␣%s", rmfile);
244
245     system(rmfile);
246
247     bzero(rmfile,sizeof(rmfile));
248
249     FILE *f = fopen(name, "ab");
250
251     printf("%s␣was␣recieved!\n", name);
252
253     fs = getFileSize(bufsocket, fs);
254
255     if (fs % bufSize > 0) {
256
257         fs = fs / bufSize + 1;
258
259     } else {
260
261         fs = fs / bufSize;
262
263     }
264
265     while (fs > 0) {
266
267         fs--;
268
```

```
269         doprocessing(bufsocket, f);
270 
271     }
272 
273     fclose(f);
274 
275     return;
276 
277 }
278 
279 
280 
281 void getUser(int sock, int bufsocket, int port, int clilen,
282 
283         struct sockaddr_in serverAddr, struct sockaddr_in
                clientAddr) {
284 
285     char user[30];
286 
287     char buffer[bufSize];
288 
289     printf("%d\n",bufsocket);
290 
291     bzero(buffer, bufSize);
292 
293     read(bufsocket, user, sizeof(user));
294 
295     chdir("/home/user/workspace/NewFileTransfer");
296 
297     int ch = chdir(user);
298 
299     if (ch == -1) {
300 
301         mkdir(user, S_IRWXU);
302 
303         chdir(user);
304 
305     }
306 
307     bzero(user, sizeof(user));
308 
309     write(bufsocket, "I␣got␣user␣name", 15);
310 
311 }
312 
313 
314 
315 void startThread(void *in){
316 
```

```
317     struct SockParams *sp = (struct SockParams*) in;
318 
319     int fs;
320 
321     char buffer[bufSize];
322 
323     char *name = buffer;
324 
325     puts("Thread was created");
326 
327     chdir("/home/user/workspace/NewFileTransfer");
328 
329        mkdir("SERVER", S_IRWXU);
330 
331        chdir("/home/user/workspace/NewFileTransfer/SERVER");
332 
333     //getUser(sp->sock, sp->bufsocket, sp->port, sp->clilen,
           sp->serverAddr, sp->clientAddr);
334 
335        while (1) {
336 
337           listenToClients(sp->sock, sp->bufsocket, sp->clilen,
                  sp->clientAddr, name, fs);
338 
339        }
340 
341 
342 
343 }
344 
345 
346 
347 int main(int argc, char *argv[]) {
348 
349     pthread_t mainthread;
350 
351     int i = 0;
352 
353     int j;
354 
355     struct SockParams sp;
356 
357     const int on = 1;
358 
359     sp.sock = socket(AF_INET, SOCK_STREAM, 0);
360 
361     if (sp.sock < 0) {
362 
363        perror("ERROR opening socket");
```

```
364
365     }
366
367     bzero((char *) &sp.serverAddr, sizeof(sp.serverAddr));
368
369     sp.port = 7777;
370
371     sp.serverAddr.sin_family = AF_INET;
372
373     sp.serverAddr.sin_addr.s_addr = INADDR_ANY;
374
375     sp.serverAddr.sin_port = htons(sp.port);
376
377     setsockopt(sp.sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(
            on));
378
379     if (bind(sp.sock, (struct sockaddr *) &sp.serverAddr,
            sizeof(sp.serverAddr)) < 0) {
380
381         perror("ERROR on binding");
382
383     }
384
385     while(1){
386
387         listen(sp.sock, 5);
388
389         sp.clilen = sizeof(sp.clientAddr);
390
391         sp.bufsocket = accept(sp.sock,(struct sockaddr *) &sp.
                clientAddr, &sp.clilen);
392
393         pthread_create(&mainthread,NULL,startThread,(void*)&sp)
                ;
394
395     }
396
397         pthread_join(mainthread,NULL);
398
399     return 0;
400
401 }
```

Linux TCP client

```
1 #include <stdio.h>
2
```

```
3  #include <sys/types.h>
4
5  #include <sys/socket.h>
6
7  #include <sys/stat.h>
8
9  #include <netinet/in.h>
10
11 #include <string.h>
12
13 #include <unistd.h>
14
15
16
17 #define bufSize 1024
18
19
20
21 int main(int argc, char *argv[]) {
22
23    int sock, port, n;
24
25    struct sockaddr_in serv_addr;
26
27    struct hostent *server;
28
29    long size;
30
31    int b;
32
33    char fileName[30], user[30];
34
35    char commandLine[15];
36
37    char buffer[bufSize];
38
39    char charSize[bufSize];
40
41    int i = 0;
42
43    int firstTime = 1;
44
45
46
47    if (argc < 3) {
48
49       fprintf(stderr, "usage␣%s␣hostname␣port\n", argv[0]);
50
51       exit(0);
```

```
52
53      }
54
55      port = atoi(argv[2]);
56
57      printf("User:");
58
59      scanf("%s", user);
60
61      sock = socket(AF_INET, SOCK_STREAM, 0);
62
63      if (sock < 0) {
64
65          perror("ERROR opening socket");
66
67          exit(1);
68
69      }
70
71      server = gethostbyname(argv[1]);
72
73      if (server == NULL) {
74
75          fprintf(stderr, "ERROR, no such host\n");
76
77          exit(0);
78
79      }
80
81      bzero((char *) &serv_addr, sizeof(serv_addr));
82
83      serv_addr.sin_family = AF_INET;
84
85      serv_addr.sin_port = htons(port);
86
87      if (connect(sock, &serv_addr, sizeof(serv_addr)) < 0) {
88
89          perror("ERROR connecting");
90
91          exit(1);
92
93      }
94
95      int ch = chdir(user);
96
97      if (ch == -1) {
98
99          mkdir(user, S_IRWXU);
100
```

```
101         chdir(user);
102 
103     }
104 
105     bzero(user, sizeof(user));
106 
107     bzero(buffer, bufSize);
108 
109     while (1) {
110 
111         scanf("%s", commandLine);
112 
113         if (strcmp(commandLine, "transfer") == 0) {
114 
115             firstTime = 1;
116 
117             printf("Enter␣file␣name:␣");
118 
119             scanf("%s", fileName);
120 
121             system("ls>ls");
122 
123             FILE *ls = fopen("ls", "r");
124 
125             int wasFile = 0;
126 
127             char charSize[30];
128 
129             char lsStr[30], newname[30];
130 
131             //sscanf(fileName, "%s\n",buffer);
132 
133             while (!feof(ls)) {
134 
135                 fgets(lsStr, sizeof(lsStr), ls);
136 
137                 int j;
138 
139                 bzero(newname,sizeof(newname));
140 
141                 for (j=0;j<30;j++){
142 
143                     if (lsStr[j]!='\n' ){
144 
145                         newname[j] = lsStr[j];
146 
147                     }
148 
149                     else break;
```

```
150
151             }
152
153             if (strcmp(newname,fileName)== 0){
154
155                 wasFile = 1;
156
157                 break;
158
159             }
160
161             bzero(lsStr, sizeof(lsStr));
162
163             bzero(newname,sizeof(newname));
164
165         }
166
167         fclose(ls);
168
169         system("rm␣ls");
170
171         if (wasFile == 0){
172
173                 puts("There␣is␣no␣such␣file!");
174
175                 continue;
176
177         }
178
179         FILE *file = fopen(fileName, "r+");
180
181         fseek(file, 0, SEEK_END);
182
183         size = ftell(file);
184
185         if (size > 1073741824){
186
187             printf("This␣file␣is␣too␣big!!!\n");
188
189             continue;
190
191         }
192
193         sprintf(charSize, "%d", size);
194
195         fseek(file, 0, SEEK_SET);
196
197         /* Create a socket point */
198
```

```
199             while (!feof(file)) {
200
201                 if (firstTime == 0) {
202
203                     b = fread(buffer, 1, sizeof(buffer), file);
204
205                 }
206
207                 if (firstTime == 1) {
208
209                     write(sock, fileName, strlen(fileName));
210
211                     firstTime = 2;
212
213                     bzero(buffer, bufSize);
214
215                     n = read(sock, buffer, bufSize - 1);
216
217                     printf("%s\n", buffer);
218
219                     if (n < 0) {
220
221                         perror("ERROR reading from socket");
222
223                         exit(1);
224
225                     }
226
227                     continue;
228
229                 } else if (firstTime == 2) {
230
231                     write(sock, charSize, strlen(charSize));
232
233                     firstTime = 0;
234
235                     bzero(buffer, bufSize);
236
237                     n = read(sock, buffer, bufSize - 1);
238
239                     printf("%s\n", buffer);
240
241                     if (n < 0) {
242
243                         perror("ERROR reading from socket");
244
245                         exit(1);
246
247                     }
```

```
248
249                 continue;
250
251
252
253             }
254
255             n = write(sock, buffer, strlen(buffer));
256
257             if (n < 0) {
258
259                 perror("ERROR␣writing␣to␣socket");
260
261                 exit(1);
262
263             }
264
265
266
267             /* Now read server response */
268
269             bzero(buffer, bufSize);
270
271             n = read(sock, buffer, bufSize - 1);
272
273             if (n < 0) {
274
275                 perror("ERROR␣reading␣from␣socket");
276
277                 exit(1);
278
279             }
280
281             printf("%s\n", buffer);
282
283         }
284
285         fclose(file);
286
287     } else if (strcmp(commandLine, "ls") == 0) {
288
289         write(sock, "ls", 2);
290
291         bzero(buffer, bufSize);
292
293         read(sock, buffer, bufSize - 1);
294
295         while (1) {
296
```

```
297             bzero(buffer, bufSize);
298
299             read(sock, buffer, bufSize - 1);
300
301             if (strcmp(buffer, "&") == 0) {
302
303                 break;
304
305             }
306
307             printf("%s", buffer);
308
309             write(sock, "o", 1);
310
311         }
312
313
314
315     } else if (strcmp(commandLine, "get") == 0) {
316
317         char filename[30];
318
319         printf("Get␣");
320
321         scanf("%s", filename);
322
323         write(sock, "get", 3);
324
325         bzero(buffer, bufSize);
326
327         read(sock, buffer, bufSize - 1);
328
329         write(sock, filename, sizeof(filename));
330
331         FILE *f = fopen(filename, "ab");
332
333         bzero(buffer, bufSize);
334
335         read(sock, buffer, bufSize - 1);
336
337         while (1) {
338
339             bzero(buffer, bufSize);
340
341             read(sock, buffer, bufSize - 1);
342
343             if (strcmp(buffer, "&") == 0) {
344
345                 fclose(f);
```

```
346
347                 break;
348
349             } else {
350
351                 fprintf(f, "%s", buffer);
352
353                 write(sock, "m", 1);
354
355             }
356
357         }
358
359
360
361     } else if (strcmp(commandLine, "mkdir") == 0) {
362
363         char dirname[30];
364
365         printf("mkdir␣");
366
367         scanf("%s", dirname);
368
369         write(sock, "mkdir", 5);
370
371         bzero(buffer, bufSize);
372
373         read(sock, buffer, bufSize - 1);
374
375         write(sock, dirname, sizeof(dirname));
376
377         bzero(buffer, bufSize);
378
379         read(sock, buffer, bufSize - 1);
380
381         bzero(buffer, bufSize);
382
383
384
385     } else if (strcmp(commandLine, "cd") == 0) {
386
387         char dirname[30];
388
389         printf("cd␣");
390
391         scanf("%s", dirname);
392
393         write(sock, "cd", 2);
394
```

```
395             bzero(buffer, bufSize);
396
397             read(sock, buffer, bufSize - 1);
398
399             write(sock, dirname, sizeof(dirname));
400
401             bzero(buffer, bufSize);
402
403             read(sock, buffer, bufSize - 1);
404
405             bzero(buffer, bufSize);
406
407         } else if (strcmp(commandLine, "close") == 0) {
408
409             return 0;
410
411         }
412
413     }
414
415     return 0;
416
417 }
```

Linux UDP server

```
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 <sys/stat.h>
7  #include <pthread.h>
8
9  #define BUFLEN 4096  //Max length of buffer
10 #define PORT 8888   //The port on which to listen for
       incoming data
11
12 struct mySocket {
13     struct sockaddr_in si_me, si_other;
14     int sock, slen, recv_len;
15     char buf[BUFLEN];
16
17 };
18
19
20 char* get3sym(char* str,char *symbols){
```

```
21    bzero(symbols,sizeof(symbols));
22    int i;
23    for (i=0;i<3;i++){
24       symbols[i]=str[i];
25    }
26    return symbols;
27 }
28
29 char* getLastSyms(char *str,char*symbols){
30    //bzero(symbols,sizeof(symbols));
31    //puts(str);
32    int i;
33    for (i=0;i<4093;i++){
34          symbols[i]=str[i+3];
35    }
36    return symbols;
37 }
38
39 void die(char *s) {
40    perror(s);
41    exit(1);
42 }
43
44 char* getFirstMessage(struct mySocket s) {
45    if ((s.recv_len = recvfrom(s.sock, s.buf, BUFLEN, 0,
46          (struct sockaddr *) &s.si_other, &s.slen)) == -1) {
47       die("recvfrom()");
48    }
49    printf("Received␣packet␣from␣%s:%d\n", inet_ntoa(s.
          si_other.sin_addr), ntohs(s.si_other.sin_port));
50    if (sendto(s.sock, s.buf, s.recv_len, 0, (struct sockaddr
          *) &s.si_other,
51          s.slen) == -1) {
52       die("sendto()");
53    }
54    return s.buf;
55 }
56
57 int main(void) {
58    struct mySocket s;
59    int i;
60    s.slen = sizeof(s.si_other);
61    char *command = s.buf;
62    if ((s.sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) ==
          -1) {
63       die("socket");
64    }
65    chdir("/home/user/workspace/UDPSERV");
66          mkdir("SERVER", S_IRWXU);
```

```
67          chdir("/home/user/workspace/UDPSERV/SERVER");
68
69      memset((char *) &s.si_me, 0, sizeof(s.si_me));
70
71      s.si_me.sin_family = AF_INET;
72      s.si_me.sin_port = htons(PORT);
73      s.si_me.sin_addr.s_addr = htonl(INADDR_ANY);
74
75      if (bind(s.sock, (struct sockaddr*) &s.si_me, sizeof(s.
            si_me)) == -1) {
76          die("bind");
77      }
78      while (1) {
79          bzero(s.buf, sizeof(s.buf));
80          command = getFirstMessage(s);
81          if (strcmp(command, "transfer") == 0) {
82              puts("transfer");
83              int count = 0;
84              recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
85                      &s.slen);
86              sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                    &s.si_other,
87                      s.slen);
88              printf("%s\n", s.buf);
89              FILE *f = fopen(s.buf, "ab");
90              bzero(s.buf, sizeof(s.buf));
91              char symbols[3], partOfBuf[BUFLEN-3];
92              while (strcmp(s.buf, "&") != 0) {
93                  count++;
94                  recvfrom(s.sock, s.buf, BUFLEN, 0,
95                          (struct sockaddr *) &s.si_other, &s.slen);
96                  get3sym(s.buf,symbols);
97                  int newdigit = atoi(symbols);
98                  if (count == newdigit){
99                      printf("Package␣number␣%s␣was␣received\n",
                            get3sym(s.buf,symbols));
100                     getLastSyms(s.buf,partOfBuf);
101                     sendto(s.sock, "ALRIGHT", BUFLEN-3, 0, (struct
                            sockaddr*) &s.si_other,
102                             s.slen);
103                     if (strcmp(s.buf, "&") != 0) {
104                         fprintf(f, "%s", getLastSyms(s.buf,
                                partOfBuf));
105                     }
106                 }
107                 else{
108                     count--;
```

```
109                 sendto(s.sock, "ERROR", BUFLEN-3, 0, (struct
                        sockaddr*) &s.si_other,
110                                 s.slen);
111                 continue;
112             }
113 
114         }
115         fclose(f);
116     } else if (strcmp(command, "ls") == 0) {
117         system("ls>ls");
118         FILE *ls = fopen("ls", "r");
119         char charSize[30];
120         char lsStr[30], buffer[30];
121         while (!feof(ls)) {
122             fgets(lsStr, sizeof(lsStr), ls);
123             if (strcmp(lsStr, "ls\n") == 0) {
124                 continue;
125             }
126             recvfrom(s.sock, s.buf, BUFLEN, 0,
127                     (struct sockaddr *) &s.si_other, &s.slen);
128             sendto(s.sock, lsStr, BUFLEN, 0, (struct sockaddr
                        *) &s.si_other,
129                     s.slen);
130             recvfrom(s.sock, s.buf, BUFLEN, 0,
131                     (struct sockaddr *) &s.si_other, &s.slen);
132             sendto(s.sock, lsStr, BUFLEN, 0, (struct sockaddr
                        *) &s.si_other,
133                     s.slen);
134             bzero(lsStr, sizeof(lsStr));
135         }
136         sendto(s.sock, "&", BUFLEN, 0, (struct sockaddr*) &s
                .si_other,
137                 s.slen);
138         recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                *) &s.si_other,
139                 &s.slen);
140         fclose(ls);
141         system("rm␣ls");
142     } else if (strcmp(command, "get") == 0) {
143         recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                *) &s.si_other,
144                 &s.slen);
145         sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                &s.si_other,
146                 s.slen);
147         FILE *f = fopen(s.buf, "r+");
148         while (!feof(f)) {
149             bzero(s.buf,BUFLEN);
150             recvfrom(s.sock, s.buf, BUFLEN, 0,
```

```
151                     (struct sockaddr *) &s.si_other, &s.slen);
152             fread(s.buf,1, sizeof(s.buf), f);
153             sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
154                     s.slen);
155         }
156         recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                *) &s.si_other,
157                 &s.slen);
158         sendto(s.sock, "&", BUFLEN, 0, (struct sockaddr*) &s
                .si_other,
159                 s.slen);
160         bzero(s.buf,BUFLEN);
161         fclose(f);
162       } else if (strcmp(command, "cd") == 0) {
163         recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                *) &s.si_other,
164                         &s.slen);
165         sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                &s.si_other,
166                         s.slen);
167         chdir(s.buf);
168         bzero(s.buf,BUFLEN);
169       } else if (strcmp(command, "mkdir") == 0) {
170         recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                *) &s.si_other,
171                         &s.slen);
172         sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                &s.si_other,
173                         s.slen);
174         printf("Directory␣%s␣was␣created\n", s.buf);
175         mkdir(s.buf, S_IRWXU);
176         bzero(s.buf,BUFLEN);
177       } else if (strcmp(command, "close") == 0) {
178         exit(1);
179       }
180
181     }
182     close(s.sock);
183     return 0;
184 }
```

Linux UDP client

```
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 <sys/types.h>
7  #include <sys/stat.h>
8
9  #define SERVER "127.0.0.1"
10 //#define SERVER "192.168.56.1"
11 #define BUFLEN 4096  //Max length of buffer
12 #define PORT 8888   //The port on which to send data
13
14 void die(char *s) {
15    perror(s);
16    exit(1);
17 }
18
19 int main(void) {
20    struct sockaddr_in si_other;
21    int sock, slen = sizeof(si_other);
22    char message[BUFLEN];
23    char user[30];
24    printf("User:");
25    scanf("%s", user);
26    int ch = chdir(user);
27    if (ch == -1) {
28       mkdir(user, S_IRWXU);
29       chdir(user);
30    }
31    bzero(user, sizeof(user));
32    if ((sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) ==
          -1) {
33       die("socket");
34    }
35
36    memset((char *) &si_other, 0, sizeof(si_other));
37    si_other.sin_family = AF_INET;
38    si_other.sin_port = htons(PORT);
39
40    if (inet_aton(SERVER, &si_other.sin_addr) == 0) {
41       fprintf(stderr, "inet_aton()␣failed\n");
42       exit(1);
43    }
44
45    while (1) {
46       bzero(message, sizeof(message));
47       scanf("%s", message);
48
49       if (sendto(sock, message, strlen(message), 0,
50             (struct sockaddr *) &si_other, slen) == -1) {
51          die("sendto()");
```

```
52          }
53
54          bzero(message, sizeof(message));
55          if (recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                *) &si_other,
56                  &slen) == -1) {
57              die("recvfrom()");
58          }
59          if (strcmp(message, "transfer") == 0) {
60              bzero(message, sizeof(message));
61              scanf("%s", message);
62              system("ls>ls");
63              FILE *ls = fopen("ls", "r");
64              int wasFile = 0;
65              char charSize[30];
66              char lsStr[30], newname[30];
67              while (!feof(ls)) {
68                  fgets(lsStr, sizeof(lsStr), ls);
69                  int j;
70                  bzero(newname, sizeof(newname));
71                  for (j = 0; j < 30; j++) {
72                      if (lsStr[j] != '\n') {
73                          newname[j] = lsStr[j];
74                      } else
75                          break;
76                  }
77                  if (strcmp(newname, message) == 0) {
78                      wasFile = 1;
79                      break;
80                  }
81                  bzero(lsStr, sizeof(lsStr));
82                  bzero(newname, sizeof(newname));
83              }
84              fclose(ls);
85              system("rm␣ls");
86              if (wasFile == 0) {
87                  puts("There␣is␣no␣such␣file!");
88                  continue;
89              }
90              sendto(sock, message, strlen(message), 0,
91                      (struct sockaddr *) &si_other, slen);
92              recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
93                      &slen);
94              FILE *file = fopen(message, "r+");
95              int count = 0;
96              char packet[BUFLEN-1];
97              char countChar[3];
98              while (!feof(file)) {
```

```
99              bzero(message, sizeof(message));
100             count++;
101             sprintf(countChar,"%d",count);
102             fread(packet, 1, sizeof(packet), file);
103             strncat(message,countChar,3);
104             if(count<10){
105                strncat(message,"␣␣",2);
106             }
107             else if(count <100){
108                strncat(message,"␣",1);
109             }
110             printf("%s\n",message);
111             strncat(message,packet,BUFLEN-4);
112             sendto(sock, message, strlen(message), 0,
113                    (struct sockaddr *) &si_other, slen);
114             bzero(message,BUFLEN);
115             recvfrom(sock, message, BUFLEN, 0,
116                    (struct sockaddr *) &si_other, &slen);
117             if (strcmp(message,"ERROR")==0){
118                fseek(file,count*BUFLEN,SEEK_SET);
119                count--;
120             }
121          }
122          fclose(file);
123          sendto(sock, "&", BUFLEN, 0, (struct sockaddr *) &
                si_other, slen);
124          recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                *) &si_other,
125                &slen);
126       }
127       if (strcmp(message, "ls") == 0) {
128          while (strcmp(message, "&") != 0) {
129             bzero(message, sizeof(message));
130             sendto(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
131                   slen);
132             recvfrom(sock, message, BUFLEN, 0,
133                    (struct sockaddr *) &si_other, &slen);
134             if (strcmp(message, "&") != 0)
135                printf("%s", message);
136             sendto(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
137                   slen);
138             recvfrom(sock, message, BUFLEN, 0,
139                    (struct sockaddr *) &si_other, &slen);
140          }
141       }
142       if (strcmp(message, "get") == 0) {
143          bzero(message, sizeof(message));
```

```
144             printf("get␣");
145             scanf("%s", message);
146             sendto(sock, message, BUFLEN, 0, (struct sockaddr *)
                    &si_other,
147                    slen);
148             FILE *f = fopen(message, "ab");
149             recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
150                    &slen);
151             while (strcmp(message, "&") != 0) {
152                 sscanf("message", "%s", message);
153                 sendto(sock, message, BUFLEN, 0, (struct sockaddr
                        *) &si_other,
154                        slen);
155                 recvfrom(sock, message, BUFLEN, 0,
156                        (struct sockaddr *) &si_other, &slen);
157                 if (strcmp(message, "&") != 0) {
158                     fprintf(f, "%s", message);
159                 } else
160                     break;
161                 bzero(message, sizeof(message));
162             }
163             fclose(f);
164         }
165         if (strcmp(message, "cd") == 0) {
166             printf("cd␣");
167             scanf("%s", message);
168             sendto(sock, message, BUFLEN, 0, (struct sockaddr *)
                    &si_other,
169                    slen);
170             recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
171                    &slen);
172             bzero(message, sizeof(message));
173         }
174         if (strcmp(message, "mkdir") == 0) {
175             printf("mkdir␣");
176             scanf("%s", message);
177             sendto(sock, message, BUFLEN, 0, (struct sockaddr *)
                    &si_other,
178                    slen);
179             recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
180                    &slen);
181             bzero(message, sizeof(message));
182         }
183         if (strcmp(message, "close") == 0) {
184             return 0;
185         }
```

```
186     }
187
188     close(sock);
189     return 0;
190 }
```

Windows TCP server

```
1  /*
2   * Server.c
3   *
4   *  Created on: Oct 16, 2014
5   *      Author: Baratynskiy
6   */
7
8  #include <io.h>
9  #include <stdio.h>
10 #include <winsock2.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <direct.h>
14 #include <locale.h>
15 #include <sys/types.h>
16 #include <limits.h>
17 #include <sys/stat.h>
18 #pragma comment(lib, "WS2_32.lib")
19 #define bufSize 1024
20 HANDLE h1;
21
22
23 struct SockParams{
24     WSADATA wsa;
25     SOCKET sock, bufsocket;
26     int port, clilen;
27     struct sockaddr_in serverAddr, clientAddr;
28 };
29
30 int doprocessing(int sock, FILE *f) {
31     char buffer[bufSize];
32     memset(buffer,0,bufSize);
33     recv(sock, buffer, bufSize - 1,0);
34     send(sock, "I got your message", 18,0);
35     fprintf(f, "%s", buffer);
36     return 0;
37 }
38
39 char* getFirstMessage(int sock, char *firstMessage) {
```

```
40      memset(firstMessage,0,bufSize);
41      recv(sock, firstMessage, bufSize - 1,0);
42      if (strcmp(firstMessage, "ls") == 0) {
43          send(sock, "ls command", 10,0);
44          firstMessage = "ls";
45          return "ls";
46      } else if (strcmp(firstMessage, "get") == 0) {
47          send(sock, "get command", 11,0);
48          firstMessage = "get";
49          return "get";
50      } else if (strcmp(firstMessage, "mkdir") == 0) {
51          send(sock, "mkdir command", 11,0);
52          firstMessage = "mkdir";
53          return "mkdir";
54      } else if (strcmp(firstMessage, "cd") == 0) {
55          send(sock, "cd command", 11,0);
56          firstMessage = "cd";
57          return "cd";
58      } else {
59          send(sock, "I got file name", 15,0);
60      }
61      return firstMessage;
62  }
63
64  int getFileSize(int sock, int size) {
65      char buffer[bufSize];
66      memset(buffer,0,bufSize);
67      recv(sock, buffer, bufSize - 1,0);
68      send(sock, "I got file size", 15,0);
69      sscanf(buffer, "%d", &size);
70      return size;
71  }
72
73  void listenToClients(int sock, int bufsocket, int clilen,
74          struct sockaddr_in clientAddr, char *name, int fs) {
75      getFirstMessage(bufsocket, name);
76      if (strcmp(name, "ls") == 0) {
77          system("dir>ls");
78          FILE *ls = fopen("ls", "r");
79          char charSize[30];
80          char lsStr[30], buffer[30];
81          while (!feof(ls)) {
82              fgets(lsStr, sizeof(lsStr), ls);
83              if (strcmp(lsStr, "ls\n") == 0 ) {
84                  continue;
85              }
86              send(bufsocket, lsStr, sizeof(lsStr),0);
87              recv(bufsocket, buffer, 1,0);
88              memset(lsStr,0,sizeof(lsStr));
```

```
89          }
90          send(bufsocket, "&", 2,0);
91          fclose(ls);
92          system("del␣ls");
93          return;
94      }
95      else if (strcmp(name, "get") == 0) {
96          char filename[30];
97          char buffer[1024];
98          recv(bufsocket, filename, sizeof(filename),0);
99          send(bufsocket, filename, sizeof(filename),0);
100         printf("%s␣is␣wanted\n", filename);
101         FILE *file = fopen(filename, "r+");
102         while (!feof(file)) {
103             fread(buffer, 1, sizeof(buffer), file);
104             send(bufsocket, buffer, sizeof(buffer),0);
105             recv(bufsocket, buffer, sizeof(buffer),0);
106         }
107         send(bufsocket, "&", 1,0);
108         recv(bufsocket, buffer, sizeof(buffer),0);
109         fclose(file);
110         return;
111     }
112     else if (strcmp(name, "mkdir") == 0) {
113         char dirname[30];
114         recv(bufsocket, dirname, sizeof(dirname),0);
115         send(bufsocket, dirname, sizeof(dirname),0);
116         printf("Directory␣%s␣was␣created\n", dirname);
117         _mkdir(dirname);
118         return;
119     }
120     else if (strcmp(name, "cd") == 0) {
121         char dirname[30];
122         recv(bufsocket, dirname, sizeof(dirname),0);
123         send(bufsocket, dirname, sizeof(dirname),0);
124         _chdir(dirname);
125         return;
126     }
127     char rmfile[70];
128     memset(rmfile,0,sizeof(rmfile));
129     sscanf(name, "del␣%s", rmfile);
130     system(rmfile);
131     memset(rmfile,0,sizeof(rmfile));
132     FILE *f = fopen(name, "ab");
133     printf("%s␣was␣recieved!\n", name);
134     fs = getFileSize(bufsocket, fs);
135     if (fs % bufSize > 0) {
136         fs = fs / bufSize + 1;
137     } else {
```

```
138         fs = fs / bufSize;
139      }
140      while (fs > 0) {
141         fs--;
142         doprocessing(bufsocket, f);
143      }
144      fclose(f);
145      return;
146   }
147
148   void getUser(int sock, int bufsocket, int port, int clilen,
149         struct sockaddr_in serverAddr, struct sockaddr_in
              clientAddr) {
150      char user[30];
151      char buffer[bufSize];
152      printf("%d\n",bufsocket);
153      memset(buffer,0,bufSize);
154      recv(bufsocket, user, sizeof(user),0);
155      _chdir("SERVER");
156      int ch = _chdir(user);
157      if (ch == -1) {
158         _mkdir(user);
159         _chdir(user);
160      }
161      memset(user,0,sizeof(user));
162      send(bufsocket, "I got user name", 15,0);
163   }
164
165   DWORD WINAPI startThread(void *in){
166      struct SockParams *sp = (struct SockParams*) in;
167      int fs=0;
168      char buffer[bufSize];
169      char *name = buffer;
170      puts("Thread was created");
171      //getUser(sp->sock, sp->bufsocket, sp->port, sp->clilen,
             sp->serverAddr, sp->clientAddr);
172         while (1) {
173            listenToClients(sp->sock, sp->bufsocket, sp->clilen,
                  sp->clientAddr, name, fs);
174         }
175
176   }
177
178   int main(int argc, char *argv[]) {
179      //pthread_t mainthread;
180      int i = 0;
181      int j;
182      struct SockParams sp;
183      const int on = 1;
```

```
184     if (WSAStartup(MAKEWORD(2,2),&sp.wsa) != 0)
185      {
186          printf("Failed. Error Code : %d",WSAGetLastError());
187          return 1;
188      }
189         //Create a socket
190      if((sp.sock = socket(AF_INET , SOCK_STREAM , 0 )) ==
            INVALID_SOCKET)
191      {
192          printf("Could not create socket : %d" ,
                WSAGetLastError());
193      }
194     memset((char *) &sp.serverAddr,0, sizeof(sp.serverAddr));
195     sp.port = 7777;
196     sp.serverAddr.sin_family = AF_INET;
197     sp.serverAddr.sin_addr.s_addr = INADDR_ANY;
198     sp.serverAddr.sin_port = htons(sp.port);
199     if (bind(sp.sock, (struct sockaddr *) &sp.serverAddr,
            sizeof(sp.serverAddr)) == SOCKET_ERROR) {
200        printf("Bind failed with error code : %d" ,
              WSAGetLastError());
201     }
202     _mkdir("SERVER");
203     _chdir("SERVER");
204     while(1){
205        listen(sp.sock, 5);
206        sp.clilen = sizeof(sp.clientAddr);
207        sp.bufsocket = accept(sp.sock,(struct sockaddr *) &sp.
              clientAddr, &sp.clilen);
208        if (sp.bufsocket == INVALID_SOCKET)
209        {
210           printf("accept failed with error code : %d" ,
                 WSAGetLastError());
211         }
212        h1 = CreateThread(NULL, 0, startThread, (void*)&sp, 0,
              NULL);
213        startThread((void*)&sp);
214     }
215
216      closesocket(sp.sock);
217     CloseHandle(h1);
218      WSACleanup();
219     return 0;
220 }
```

Windows TCP client

```c
#include <io.h>
#include <stdio.h>
#include <winsock2.h>
#include <stdlib.h>
#include <string.h>
#include <direct.h>
#include <locale.h>
#include <sys/types.h>
#include <limits.h>
#include <sys/stat.h>
#pragma comment(lib, "WS2_32.lib")

#define bufSize 1024

int main(int argc, char *argv[]) {
   WSADATA wsa;
   SOCKET sock;
   int port, n;
   struct sockaddr_in serv_addr;
   long size;
   int b;
   char fileName[30], user[30];
   char commandLine[15];
   char buffer[bufSize];
   char charSize[bufSize];
   int i = 0;
   int firstTime = 1;

   if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }
   printf("User:");
   scanf("%s", user);
   if((sock = socket(AF_INET , SOCK_STREAM , 0 )) ==
      INVALID_SOCKET)
    {
        printf("Could not create socket : %d" ,
           WSAGetLastError());
    }
   memset((char *) &serv_addr,0, sizeof(serv_addr));
   serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(7777);
   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(
      serv_addr)) < 0) {
      perror("ERROR connecting");
      exit(1);
```

```
47      }
48      int ch = _chdir(user);
49      if (ch == -1) {
50          _mkdir(user);
51          _chdir(user);
52      }
53      memset(user, 0,sizeof(user));
54      memset(buffer,0, bufSize);
55      while (1) {
56          scanf("%s", commandLine);
57          if (strcmp(commandLine, "transfer") == 0) {
58              firstTime = 1;
59              printf("Enter␣file␣name:␣");
60              scanf("%s", fileName);
61              system("dir␣/B>ls");
62              FILE *ls = fopen("ls", "r");
63              int wasFile = 0;
64              char charSize[30];
65              char lsStr[30], newname[30];
66              //sscanf(fileName, "%s\n",buffer);
67              while (!feof(ls)) {
68                  fgets(lsStr, sizeof(lsStr), ls);
69                  int j;
70                  memset(newname,0,sizeof(newname));
71                  for (j=0;j<30;j++){
72                      if (lsStr[j]!='\n' ){
73                          newname[j] = lsStr[j];
74                      }
75                      else break;
76                  }
77                  if (strcmp(newname,fileName)== 0){
78                      wasFile = 1;
79                      break;
80                  }
81                  memset(lsStr, 0, sizeof(lsStr));
82                  memset(newname,0, sizeof(newname));
83              }
84              fclose(ls);
85              system("del␣ls");
86              if (wasFile == 0){
87                      puts("There␣is␣no␣such␣file!");
88                      continue;
89              }
90              FILE *file = fopen(fileName, "r+");
91              fseek(file, 0, SEEK_END);
92              size = ftell(file);
93              if (size > 1073741824){
94                  printf("This␣file␣is␣too␣big!!!\n");
95                  continue;
```

```
96          }
97          sprintf(charSize, "%d", size);
98          fseek(file, 0, SEEK_SET);
99          /* Create a socket point */
100         while (!feof(file)) {
101            if (firstTime == 0) {
102               b = fread(buffer, 1, sizeof(buffer), file);
103            }
104            if (firstTime == 1) {
105               send(sock, fileName, strlen(fileName),0);
106               firstTime = 2;
107               memset(buffer,0, bufSize);
108               recv(sock, buffer, bufSize - 1,0);
109               printf("%s\n", buffer);
110               /*if (n < 0) {
111                  perror("ERROR recving from socket");
112                  exit(1);
113               }*/
114               continue;
115            } else if (firstTime == 2) {
116               send(sock, charSize, strlen(charSize),0);
117               firstTime = 0;
118               memset(buffer,0, bufSize);
119               recv(sock, buffer, bufSize - 1,0);
120               printf("%s\n", buffer);
121               /*if (n < 0) {
122                  perror("ERROR recving from socket");
123                  exit(1);
124               }*/
125               continue;
126
127            }
128            n = send(sock, buffer, strlen(buffer),0);
129            if (n < 0) {
130               perror("ERROR writing to socket");
131               exit(1);
132            }
133
134            /* Now recv server response */
135            memset(buffer,0, bufSize);
136            n = recv(sock, buffer, bufSize - 1,0);
137            if (n < 0) {
138               perror("ERROR recving from socket");
139               exit(1);
140            }
141            printf("%s\n", buffer);
142         }
143         fclose(file);
144      } else if (strcmp(commandLine, "ls") == 0) {
```

```
145             send(sock, "ls", 2,0);
146             memset(buffer,0, bufSize);
147             recv(sock, buffer, bufSize - 1,0);
148             while (1) {
149                 memset(buffer,0, bufSize);
150                 recv(sock, buffer, bufSize - 1,0);
151                 if (strcmp(buffer, "&") == 0) {
152                     break;
153                 }
154                 printf("%s", buffer);
155                 send(sock, "o", 1,0);
156             }
157 
158         } else if (strcmp(commandLine, "get") == 0) {
159             char filename[30];
160             printf("Get␣");
161             scanf("%s", filename);
162             send(sock, "get", 3,0);
163             memset(buffer,0, bufSize);
164             recv(sock, buffer, bufSize - 1,0);
165             send(sock, filename, sizeof(filename),0);
166             FILE *f = fopen(filename, "ab");
167             memset(buffer,0, bufSize);
168             recv(sock, buffer, bufSize - 1,0);
169             while (1) {
170                 memset(buffer,0, bufSize);
171                 recv(sock, buffer, bufSize - 1,0);
172                 if (strcmp(buffer, "&") == 0) {
173                     fclose(f);
174                     break;
175                 } else {
176                     fprintf(f, "%s", buffer);
177                     send(sock, "m", 1,0);
178                 }
179             }
180 
181         } else if (strcmp(commandLine, "_mkdir") == 0) {
182             char dirname[30];
183             printf("_mkdir␣");
184             scanf("%s", dirname);
185             send(sock, "_mkdir", 5,0);
186             memset(buffer, 0,bufSize);
187             recv(sock, buffer, bufSize - 1,0);
188             send(sock, dirname, sizeof(dirname),0);
189             memset(buffer,0, bufSize);
190             recv(sock, buffer, bufSize - 1,0);
191             memset(buffer,0, bufSize);
192 
193         } else if (strcmp(commandLine, "cd") == 0) {
```

```
194             char dirname[30];
195             printf("cd␣");
196             scanf("%s", dirname);
197             send(sock, "cd", 2,0);
198             memset(buffer,0, bufSize);
199             recv(sock, buffer, bufSize - 1,0);
200             send(sock, dirname, sizeof(dirname),0);
201             memset(buffer,0, bufSize);
202             recv(sock, buffer, bufSize - 1,0);
203             memset(buffer,0, bufSize);
204         } else if (strcmp(commandLine, "close") == 0) {
205             return 0;
206         }
207     }
208     return 0;
209 }
```

Windows UDP server

```
1  #include <io.h>
2  #include <stdio.h>
3  #include <winsock2.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <direct.h>
7  #include <locale.h>
8  #include <sys/types.h>
9  #include <limits.h>
10 #include <sys/stat.h>
11 #pragma comment(lib, "WS2_32.lib")
12
13 #define BUFLEN 1024  //Max length of buffer
14 #define PORT 8888   //The port on which to listen for
       incoming data
15
16 struct mySocket {
17     struct sockaddr_in si_me, si_other;
18     SOCKET sock;
19     int slen, recv_len;
20     char buf[BUFLEN];
21     WSADATA wsa;
22 };
23
24 void die(char *s) {
25     perror(s);
26     exit(1);
27 }
```

```
28
29 char* getFirstMessage(struct mySocket s) {
30    if ((s.recv_len = recvfrom(s.sock, s.buf, BUFLEN, 0,
31          (struct sockaddr *) &s.si_other, &s.slen)) == -1) {
32       die("recvfrom()");
33    }
34    printf("Received␣packet␣from␣%s:%d\n", inet_ntoa(s.
         si_other.sin_addr), ntohs(s.si_other.sin_port));
35    if (sendto(s.sock, s.buf, s.recv_len, 0, (struct sockaddr
         *) &s.si_other,
36          s.slen) == -1) {
37       die("sendto()");
38    }
39    return s.buf;
40 }
41
42 int main(void) {
43    struct mySocket s;
44    int i;
45       if (WSAStartup(MAKEWORD(2,2),&s.wsa) != 0)
46     {
47         printf("Failed.␣Error␣Code␣:␣%d",WSAGetLastError());
48         exit(EXIT_FAILURE);
49     }
50    s.slen = sizeof(s.si_other);
51    char *command = s.buf;
52
53    if ((s.sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) ==
          INVALID_SOCKET) {
54        printf("Could␣not␣create␣socket␣:␣%d" ,
             WSAGetLastError());
55    }
56
57          _mkdir("SERVER");
58          _chdir("SERVER");
59
60    memset((char *) &s.si_me, 0, sizeof(s.si_me));
61
62    s.si_me.sin_family = AF_INET;
63    s.si_me.sin_port = htons(PORT);
64    s.si_me.sin_addr.s_addr = htonl(INADDR_ANY);
65
66    if (bind(s.sock, (struct sockaddr*) &s.si_me, sizeof(s.
         si_me)) == SOCKET_ERROR) {
67       die("bind");
68    }
69    while (1) {
70       memset(s.buf, 0, sizeof(s.buf));
71       command = getFirstMessage(s);
```

```
72          if (strcmp(command, "transfer") == 0) {
73              puts("transfer");
74              recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
75                      &s.slen);
76              sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                    &s.si_other,
77                      s.slen);
78              printf("%s\n", s.buf);
79              FILE *f = fopen(s.buf, "ab");
80              memset(s.buf,0, sizeof(s.buf));
81              while (strcmp(s.buf, "&") != 0) {
82                  recvfrom(s.sock, s.buf, BUFLEN, 0,
83                          (struct sockaddr *) &s.si_other, &s.slen);
84                  sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                        *) &s.si_other,
85                          s.slen);
86                  if (strcmp(s.buf, "&") != 0) {
87                      fprintf(f, "%s", s.buf);
88                  }
89              }
90              fclose(f);
91          } else if (strcmp(command, "ls") == 0) {
92              system("dir>ls");
93              FILE *ls = fopen("ls", "r");
94              char charSize[30];
95              char lsStr[30], buffer[30];
96              while (!feof(ls)) {
97                  fgets(lsStr, sizeof(lsStr), ls);
98                  if (strcmp(lsStr, "ls\n") == 0) {
99                      continue;
100                 }
101                 recvfrom(s.sock, s.buf, BUFLEN, 0,
102                         (struct sockaddr *) &s.si_other, &s.slen);
103                 sendto(s.sock, lsStr, BUFLEN, 0, (struct sockaddr
                        *) &s.si_other,
104                         s.slen);
105                 recvfrom(s.sock, s.buf, BUFLEN, 0,
106                         (struct sockaddr *) &s.si_other, &s.slen);
107                 sendto(s.sock, lsStr, BUFLEN, 0, (struct sockaddr
                        *) &s.si_other,
108                         s.slen);
109                 memset(lsStr,0, sizeof(lsStr));
110             }
111             sendto(s.sock, "&", BUFLEN, 0, (struct sockaddr*) &s
                    .si_other,
112                     s.slen);
113             recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
```

```
114                 &s.slen);
115             fclose(ls);
116             system("del␣ls");
117         } else if (strcmp(command, "get") == 0) {
118             recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
119                 &s.slen);
120             sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                    &s.si_other,
121                 s.slen);
122             FILE *f = fopen(s.buf, "r+");
123             while (!feof(f)) {
124                 memset(s.buf,0,BUFLEN);
125                 recvfrom(s.sock, s.buf, BUFLEN, 0,
126                       (struct sockaddr *) &s.si_other, &s.slen);
127                 fread(s.buf,1, sizeof(s.buf), f);
128                 sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                        *) &s.si_other,
129                     s.slen);
130             }
131             recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
132                 &s.slen);
133             sendto(s.sock, "&", BUFLEN, 0, (struct sockaddr*) &s
                    .si_other,
134                 s.slen);
135             memset(s.buf,0,BUFLEN);
136             fclose(f);
137         } else if (strcmp(command, "cd") == 0) {
138             recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
139                         &s.slen);
140             sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                    &s.si_other,
141                         s.slen);
142             chdir(s.buf);
143             memset(s.buf,0,BUFLEN);
144         } else if (strcmp(command, "mkdir") == 0) {
145             recvfrom(s.sock, s.buf, BUFLEN, 0, (struct sockaddr
                    *) &s.si_other,
146                         &s.slen);
147             sendto(s.sock, s.buf, BUFLEN, 0, (struct sockaddr*)
                    &s.si_other,
148                         s.slen);
149             printf("Directory␣%s␣was␣created\n", s.buf);
150             _mkdir(s.buf);
151             memset(s.buf,0,BUFLEN);
152         } else if (strcmp(command, "close") == 0) {
153             exit(1);
```

```
154         }
155
156     }
157     close(s.sock);
158     return 0;
159 }
```

Windows UDP client

```
1  #include <io.h>
2  #include <stdio.h>
3  #include <winsock2.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <direct.h>
7  #include <locale.h>
8  #include <sys/types.h>
9  #include <limits.h>
10 #include <sys/stat.h>
11 #pragma comment(lib, "WS2_32.lib")
12 #define SERVER "127.0.0.1"
13 #define BUFLEN 1024  //Max length of buffer
14 #define PORT 8888   //The port on which to send data
15
16 void die(char *s) {
17     perror(s);
18     exit(1);
19 }
20
21 int main(void) {
22     struct sockaddr_in si_other;
23     SOCKET sock;
24     WSADATA wsa;
25     int  slen = sizeof(si_other);
26     char message[BUFLEN];
27     char user[30];
28         if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
29      {
30          printf("Failed. Error Code : %d",WSAGetLastError());
31          exit(EXIT_FAILURE);
32      }
33     printf("User:");
34     scanf("%s", user);
35     int ch = _chdir(user);
36     if (ch == -1) {
37        _mkdir(user);
38        _chdir(user);
```

```
39      }
40      memset(user,0, sizeof(user));
41          if ( (sock=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))
                == SOCKET_ERROR)
42       {
43           printf("socket() failed with error code : %d" ,
                WSAGetLastError());
44           exit(EXIT_FAILURE);
45       }
46
47      memset((char *) &si_other, 0, sizeof(si_other));
48      si_other.sin_family = AF_INET;
49      si_other.sin_port = htons(PORT);
50       si_other.sin_addr.S_un.S_addr = inet_addr(SERVER);
51
52      while (1) {
53         memset(message,0, sizeof(message));
54         scanf("%s", message);
55
56         if (sendto(sock, message, strlen(message), 0,
57                (struct sockaddr *) &si_other, slen) == -1) {
58            die("sendto()");
59         }
60
61         memset(message,0, sizeof(message));
62         if (recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                *) &si_other,
63               &slen) == -1) {
64            die("recvfrom()");
65         }
66         if (strcmp(message, "transfer") == 0) {
67            memset(message,0, sizeof(message));
68            scanf("%s", message);
69            system("dir /B>ls");
70            FILE *ls = fopen("ls", "r");
71            int wasFile = 0;
72            char charSize[30];
73            char lsStr[30], newname[30];
74            while (!feof(ls)) {
75               fgets(lsStr, sizeof(lsStr), ls);
76               int j;
77               memset(newname,0, sizeof(newname));
78               for (j = 0; j < 30; j++) {
79                  if (lsStr[j] != '\n') {
80                     newname[j] = lsStr[j];
81                  } else
82                     break;
83               }
84               if (strcmp(newname, message) == 0) {
```

```
85                wasFile = 1;
86                break;
87             }
88             memset(lsStr, 0,sizeof(lsStr));
89             memset(newname,0, sizeof(newname));
90          }
91          fclose(ls);
92          system("del␣ls");
93          if (wasFile == 0) {
94             puts("There␣is␣no␣such␣file!");
95             continue;
96          }
97          sendto(sock, message, strlen(message), 0,
98                (struct sockaddr *) &si_other, slen);
99          recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
               *) &si_other,
100               &slen);
101         FILE *file = fopen(message, "r+");
102         while (!feof(file)) {
103            memset(message,0, sizeof(message));
104            fread(message, 1, sizeof(message), file);
105            sendto(sock, message, strlen(message), 0,
106                  (struct sockaddr *) &si_other, slen);
107            recvfrom(sock, message, BUFLEN, 0,
108                  (struct sockaddr *) &si_other, &slen);
109         }
110         fclose(file);
111         sendto(sock, "&", BUFLEN, 0, (struct sockaddr *) &
               si_other, slen);
112         recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
               *) &si_other,
113               &slen);
114      }
115      if (strcmp(message, "ls") == 0) {
116         while (strcmp(message, "&") != 0) {
117            memset(message, 0,sizeof(message));
118            sendto(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
119                  slen);
120            recvfrom(sock, message, BUFLEN, 0,
121                  (struct sockaddr *) &si_other, &slen);
122            if (strcmp(message, "&") != 0)
123               printf("%s", message);
124            sendto(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
125                  slen);
126            recvfrom(sock, message, BUFLEN, 0,
127                  (struct sockaddr *) &si_other, &slen);
128         }
```

```
129         }
130         if (strcmp(message, "get") == 0) {
131             memset(message, 0,sizeof(message));
132             printf("get␣");
133             scanf("%s", message);
134             sendto(sock, message, BUFLEN, 0, (struct sockaddr *)
                    &si_other,
135                    slen);
136             FILE *f = fopen(message, "ab");
137             recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
138                    &slen);
139             while (strcmp(message, "&") != 0) {
140                 sscanf("message", "%s", message);
141                 sendto(sock, message, BUFLEN, 0, (struct sockaddr
                        *) &si_other,
142                        slen);
143                 recvfrom(sock, message, BUFLEN, 0,
144                        (struct sockaddr *) &si_other, &slen);
145                 if (strcmp(message, "&") != 0) {
146                     fprintf(f, "%s", message);
147                 } else
148                     break;
149                 memset(message,0, sizeof(message));
150             }
151             fclose(f);
152         }
153         if (strcmp(message, "cd") == 0) {
154             printf("cd␣");
155             scanf("%s", message);
156             sendto(sock, message, BUFLEN, 0, (struct sockaddr *)
                    &si_other,
157                    slen);
158             recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
159                    &slen);
160             memset(message,0, sizeof(message));
161         }
162         if (strcmp(message, "mkdir") == 0) {
163             printf("mkdir␣");
164             scanf("%s", message);
165             sendto(sock, message, BUFLEN, 0, (struct sockaddr *)
                    &si_other,
166                    slen);
167             recvfrom(sock, message, BUFLEN, 0, (struct sockaddr
                    *) &si_other,
168                    &slen);
169             memset(message,0, sizeof(message));
170         }
```

```
171         if (strcmp(message, "close") == 0) {
172             return 0;
173         }
174     }
175 
176     close(sock);
177     return 0;
178 }
```

Пример файла сборки Makefile

```
1 all:
2 
3     gcc -g -O0 -Wall -o tcp_serv -lpthread Server.c
```