Сети ЭВМ и телекоммуникации

Д. С. Мурашко

7 января 2015 г.

Глава 1

Калькулятор

Разработать приложение-сервер «Удаленный калькулятор», позволяющее по запросу выполнять математические операции, и удаленный клиент для сервера.

1.1 Функциональные требования

Серверное приложение должно реализовывать следующие функции:

1. Приём «быстрых» операций с аргументами от клиента. Должны поддерживаться следующие операции: сложение, вычитание, умножение, деление.
2. Вычисление «долгих» математических операций (факториал, квадратный корень) с последующей отложенной посылкой результата клиенту(отдельная операция, инициируемая сервером).
3. Обработка запроса на отключение клиента
4. Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1. Посылка операции с аргументами на вычисление
2. Получение результата вычислений«быстрых» операций
3. Получения результата вычислений«долгих» операций
4. Разрыв соединения
5. 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. Ограничение на вводимые данные: результат выражения может "выйти"за пределы типа double (больше 19 разрядов), причем такая ситуация возможна для всех операций. Это является большим минусов приложения. Решить эту проблему возможно с помощью вычислений с произвольной точностью, использовав, к примеру, библиотеку GMP.
2. Ограничения на длину сообщения: максимальный размер 40 символов.Для правильного чтения/отправки сообщения требуется фиксированная длина.
3. Ограниченние, накладываемое на вычисление факториала: аргумент не должен быть больше 16 (идет переполнение long int)
4. Обрыв сессии - некорректное завершение работы клиентом. При некорректном завершении сессии клиентом он останется в состоянии "online что является минусом данного протокола

Глава 2

Реализация для работы по протоколу TCP

2.1 Прикладной протокол

Клиент и сервер обмениваются сообщениями. Сообщение от клиента – запрос, сообщение от сервера – ответ. В ходе работы клиент посылает запросы серверу. Сервер обязан отправить ответ. На каждый запрос должен быть отправлен только один ответ. В качестве сообщения передается строка с арифметическим выражением. Предусмотренны следующие команды: сложение, вычитание, умножение, деление, вычисление факториала и квадратного корня:

1. Формат для операций сложения, вычитания, умножения и деления:

   (a) Первое число

   (b) Знак операции (+, -, *, /)

   (c) Второе число

   (d) Равно (=)

   Дробная часть вводится через "."

2. Формат для операций факториал и квадратный корень:

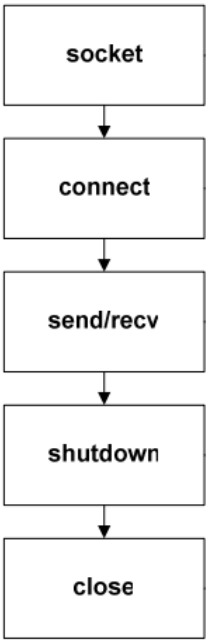
(a) Число

(b) Знак операции (!)

2.2 Архитектура приложения

2.2.1 Описание клиента

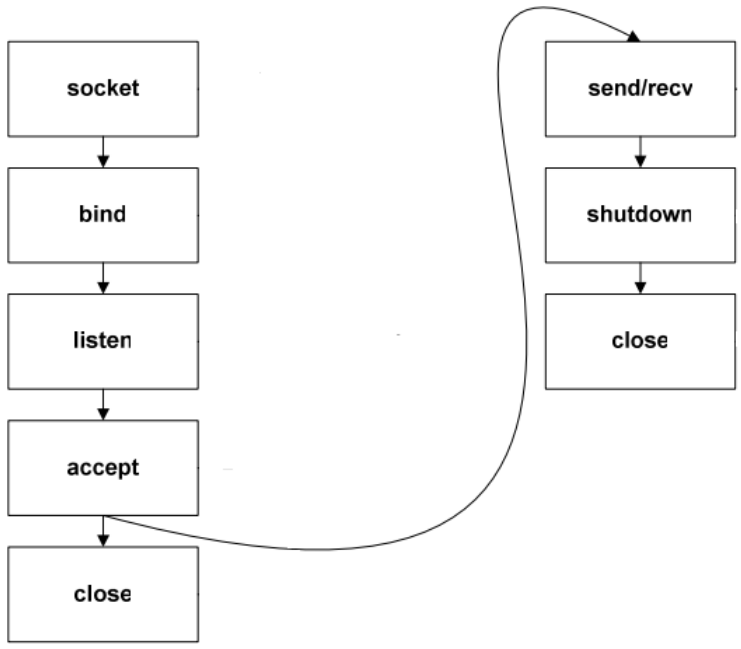
Клиент протокола TCP создаёт экземпляр сокета, необходимый для взаимодействия с сервером, организует соединение, осуществляет обмен данными, в соответствии с протоколом прикладного уровня. Структура TCP-клиента представлена ниже:



2.2.2 Описание сервера

Организация TCP-сервера отличается отTCP-клиента в первую очередь созданием слушающего сокета. Такой сокет находится в состоянии listen и предназначен только для приёма входящих соединений. В случае прихода запроса на соединение создаётся дополнительный сокет, который

и занимается обменом данными с клиентом. Типичная структура TCP-сервера и взаимосвязь сокетов изображена ниже:



а) Слушающий сокет б) Порождённый сокет

2.2.3 Дизайн приложения

После запуска сервер начинает прослушивать определенный порт (5001) и при подключении клиента, выделяет ему отдельный поток. При подключнии к серверу выводится:
Input Expression:
Далее клиент вводит выражение согласно протоколу. В случае ввода "быстрых"операций клиент сразу получает ответ в формате:
Answer = 45 и предлогается ввести следующее выражение.
Результат вычисления "долгих"математических операций (факториал, квадратный корень) посылаюся с отложенной посылкой результата клиенту. Для получения результата клиент вводит команду CHECK-F или CHECK-S для факториала или квадратного корня соответственно. Чтобы завершить соединение клиент должен ввести: q.

2.2.4 Многопоточное взаимодействие с несколькими клиентами

Для осуществления многоклиентского взаимодействия используются потоки, каждый поток защищен от другого клиента. Сервер при постоянном режиме прослушивает необходимый порт и при подключении клиента, выделяет ему отдельный поток и ждет сообщения от клиента. Поток завершается при отключении клиента. Сервер в любой момент может отключить подключившегося к нему клиента. Для этого требуется ввести команду:
k[numb],
где [numb]-номер подключившегося клиента. Сервер закроет поток, соответствующий этому клиенту, тем самым отключит выбранного клиента.

Описание среды разработки

Linux debian 3.2.0-4-486 1 Debian 3.2.60-1+deb7u3 i686 GNU/Linux.
Среда разработки - Eclipse.
Windows 7.
Среда разработки - Visual Studio 2010.

2.3 Тестирование

2.3.1 Описание тестового стенда и методики тестирования

Для тестирования приложения запускается сервер и несколько клиентов. В процессе тестирования проверяются основные возможности сервера по параллельному тестированию нескольких пользователей.

2.3.2 Тестовый план и результаты тестирования

Протесируем вначале быстрые операции: сложение, вычиание, умножение и деление. Проверим работу с положительными и отрицательными числами, деление на 0.
Input Expression:
2+2=
Answer = 4.000000

Input Expression:

5-6=
Answer = -1.000000

Input Expression:
-5+4=
Answer = -1.000000

Input Expression:
253.4678*4515=
Answer = 1144407.117000

Input Expression:
362626/57483435=
Answer = 0.006308

Input Expression:
3435/0=
Error, input divider != 0 !

Input Expression:
-56/-78=
Answer = 0.717949

Теперь проверим долгие операции: факториал и квадратный корень. Проверим работу, как поведет себя приложение при задании неверных аргументов для этих операций, а также протестируем отложенную посылку результата.
строку ввести Input Expression:
4!
Wait

Input Expression:
45-90=
Answer = -45.000000

Input Expression:
CHECK-F
Factorial = 24

Input Expression:
45

Wait

Input Expression:
57/89=
Answer = 0.640449

Input Expression:
CHECK-S
Sqrt = 6.708204

Input Expression:
-5!
Error, input Factorial > 0 !

Input Expression:
-5
Error, input Root >= 0 !

При задании достаточно большого аргумента для факториала, приложение "падает т.к. результат операции выходит из максимального значения типа long int Input Expression:
20!
Wait

Input Expression:
CHECK-F
Factorial = -2102132736
Проверим работу приожения с слишком большими числами:
Input Expression:
3456789045678945678456 7+1=
Too long operand 1!

Input Expression:
1+12345678912345678945678678=
Too long operand 2!

Проверим многопоточность - запустим несколько клиентов одновременно (проверялось 4 клиента одновременно, но максимально возможно 10). Приложение работает корректно со всеми клиентами.
Проверим независимость потоков: для этого добавим sleep(10) для одной из операций. При вычислении этой операции данный поток ждет 10

секунд, другой, не "спит работает правильно.

При вводе неверных данных (к примеру: abc), приложение также "падает т.к. данные не соответствуют протоколу.

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

Прикладной протокол UDP ничем не отличается от TCP, т.к. также передается строка с арифметическим выражением (см.2.1, описывающий протокол для взаимодействия по TCP).

3.2 Архитектура приложения и Тестирование

Архитектура приложения и тестирование такие же как у TCP.

Глава 4

Выводы

4.1 TCP

TCP – протокол с обеспечением надежности передачи. TCP гарантирует, что данные не потеряются в пути, придут в правильном порядке и не придут дважды. Однако, так как TCP – потоковый протокол, может возникнуть проблема, связанная с вызовом read/recv , т.к. он может считать лишь часть сообщения. Таким образом, для чтения одного сообщения может понадобиться несколько вызовов read/recv. Поэтому в данном приложении длина сообщения ограниченна. Также протокол TCP требует, чтобы все отправленные сегменты данных были подтверждены с приёмного конца, т.е. используется алгоритм обратной связи.

4.2 UDP

Протокол UDP называют протоколом ненадёжной доставки. 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.

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 7.
Среда разработки - Visual Studio 2010.

Листинги

TCP сервер

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <string.h>
7  #include <math.h>
8  #include <gmp.h>
9  #include <stdbool.h>
10 #define max_conn 10
11 #define max_delays_num 100
12 #define BUFLEN 40
13
14 int socks[max_conn];
15 int sockfd;
16
17 long int fact_args[max_delays_num];
18 pthread_t fact_thread[max_delays_num];
19 long int fact_res[max_delays_num];
20
21 double sqrt_args[max_delays_num];
22 pthread_t sqrt_thread[max_delays_num];
23 double sqrt_res[max_delays_num];
```

```
24
25 void* fun(void* arg){
26    int newsockfd = *(int*)arg;
27    while(1){
28       char text[60],tmp[60];
29       int kill_client=0;//нужно ли убить заданного клиента
30       int exit_server=0;
31       int numb;
32       fgets(text,59,stdin);
33       if(text[0]=='q'){
34          printf("exit\n");
35          shutdown(sockfd,2);
36          close(sockfd);
37
38       }
39       if(text[0]=='k'){
40          kill_client=1;
41          numb=text[1]-48;
42          printf("numb␣%d\n",numb);
43       }
44       else {
45          kill_client=0;
46
47       }
48       if (kill_client==1){
49          shutdown(socks[numb],2);
50          close(socks[numb]);
51          kill_client=0;
52       }
53
54    }
55 }
56 void* threads(void* arg) {
57    int newsockfd = *(int*)arg;
58    char buffer[BUFLEN], ans[BUFLEN];
59    int n;
60    while (1){
61       bzero(buffer, BUFLEN);
62       n = read(newsockfd, buffer, BUFLEN-1);
63       /*if (strlen(buffer)>30){
64          write(newsockfd, "Too long expression!\n", strlen("
               Too long expression!\n"));
65       }*/
66       //else
67       calculation(buffer, ans, n, newsockfd);
68    }
69 }
70
71 long int factorial(long int x) {
```

```
72      return !x ? 1 : x * factorial(x - 1);
73  }
74
75  void run_fact(void* args) {
76
77      int* sockfd_p = (int*) args;
78      int sockfd = *sockfd_p;
79      fact_res[sockfd] = factorial(fact_args[sockfd]);
80  }
81
82  void run_sqrt(void* args) {
83      int* sockfd_p = (int*) args;
84      int sockfd = *sockfd_p;
85      sqrt_res[sockfd] = sqrt(sqrt_args[sockfd]);
86  }
87
88  void calculation(char *buffer, char *ans, int n, int
        newsockfd) {
89      int i, j;
90      long int f,ansf;
91      double a, b, k, answer;
92      char temp1[BUFLEN],temp2[BUFLEN];
93      int size;
94      //printf("Received: %s\n", buffer);
95
96      bzero(temp1, BUFLEN);
97      bzero(temp2, BUFLEN);
98      if (strncmp(buffer, "CHECK_F", 7) == 0) {
99          if (fact_res[newsockfd] != 0) {
100
101             sprintf(temp1, "Factorial␣=␣%ld\n", fact_res[
                    newsockfd]);
102             write(newsockfd, temp1, strlen(temp1));
103             printf("factorial␣=␣%ld\n", fact_res[newsockfd]);
104             fact_res[newsockfd] = 0;
105         }
106         else {
107             write(newsockfd, "Not␣yet", strlen("Not␣yet"));
108         }
109         return;
110     }
111
112     if (strncmp(buffer, "CHECK_S", 7) == 0) {
113         if (sqrt_res[newsockfd] != 0) {
114             sprintf(temp1, "Sqrt␣=␣%lf\n", sqrt_res[newsockfd]);
115             write(newsockfd, temp1, strlen(temp1));
116             printf("sqrt␣=␣%lf\n", sqrt_res[newsockfd]);
117             sqrt_res[newsockfd] = 0;
118         }
```

```
119         else {
120             write(newsockfd, "Not␣yet", strlen("Not␣yet"));
121         }
122         return;
123     }
124 
125     for (i = 1; i < (BUFLEN-1); i++) {
126         //factorial
127         if (buffer[i] == '!'){
128             strncpy(temp1, buffer, i);
129             f = atoi(temp1);
130             break;
131         }
132         //sqrt
133         if (buffer[i] == '#'){
134             strncpy(temp1, buffer, i);
135             if (strlen(temp1)>20){
136                 write(newsockfd, "Too␣long␣operand␣1!\n", strlen(
                        "Too␣long␣operand␣1!\n"));
137             }else
138             a = atof(temp1);
139             break;
140         }
141         // *, /, +,-
142         if ((buffer[i] == '*') ||(buffer[i] == '/') ||(buffer[i
                ] == '+') || (buffer[i] == '-'))  {
143             strncpy(temp1, buffer, i);
144             if (strlen(temp1)>20){
145                 write(newsockfd, "Too␣long␣operand␣1!\n", strlen(
                        "Too␣long␣operand␣1!\n"));
146             }else
147             a = atof(temp1);
148             break;
149         }
150     }
151     //end expression
152     for (j = 0; j < (BUFLEN-1); j++) {
153         if (buffer[j] == '=') {
154             strncpy(temp2, buffer + i + 1, j - i - 1);
155             if (strlen(temp2)>20){
156                 write(newsockfd, "Too␣long␣operand␣2!\n", strlen(
                        "Too␣long␣operand␣2!\n"));
157             }else
158             b = atof(temp2);
159             break;
160         }
161     }
162 
163     if (buffer[i] == '+'){
```

```
164         answer = a+b;
165         sprintf(ans, "Answer = %lf\n", answer);
166         write(newsockfd, ans, strlen(ans));
167     }
168     if (buffer[i] == '-'){
169         answer = a-b;
170         sprintf(ans, "Answer = %lf\n", answer);
171         write(newsockfd, ans, strlen(ans));
172     }
173     if (buffer[i] == '*'){
174         answer = a*b;
175         //mpz_mul (answer,a,b);
176         sprintf(ans, "Answer = %lf\n", answer);
177         write(newsockfd, ans, strlen(ans));
178     }
179     if (buffer[i] == '/'){
180         if (b==0){
181             write(newsockfd,"Error, input divider != 0 !\n",
                    strlen("Error, input divider != 0 !\n"));
182         }
183         else {
184             answer = a/b;
185             sprintf(ans, "Answer = %lf\n", answer);
186             write(newsockfd, ans, strlen(ans));
187         }
188     }
189     if (buffer[i] == '#'){
190         if (a<0){
191             write(newsockfd,"Error, input Root >= 0 !\n",strlen(
                    "Error, input Root > 0 !\n"));
192         }
193         else {
194             sqrt_args[newsockfd] = a;
195             sqrt_res[newsockfd] = 0;
196             pthread_create(&(sqrt_thread[newsockfd]),
197                    NULL,
198                    run_sqrt,
199                    (void*) &newsockfd);
200             write(newsockfd, "Wait\n", strlen("Wait\n"));
201         }
202     }
203     if (buffer[i] == '!'){
204         if (f<=0){
205             write(newsockfd,"Error, input Factorial > 0 !\n",
                    strlen("Error, input Factorial > 0 !\n"));
206         }
207         else {
208             fact_args[newsockfd] = f;
209             fact_res[newsockfd] = 0;
```

```
210             pthread_create(&(fact_thread[newsockfd]),
211                     NULL,
212                     run_fact,
213                     (void*) &newsockfd);
214             write(newsockfd, "Wait\n", strlen("Wait\n"));
215         }
216     }
217 }
218
219 int main(int argc, char *argv[]) {
220     int newsockfd, portno, clilen;
221     char buffer[BUFLEN], ans[BUFLEN];
222     struct sockaddr_in serv_addr, cli_addr;
223     int n;
224     pthread_t worker_thread[max_conn];
225     /* First call to socket() function */
226     sockfd = socket(AF_INET, SOCK_STREAM, 0);
227     if (sockfd < 0) {
228         perror("ERROR opening socket");
229         exit(1);
230     }
231     /* Initialize socket structure */
232     bzero((char *) &serv_addr, sizeof(serv_addr));
233     portno = 5001;
234     serv_addr.sin_family = AF_INET;
235     serv_addr.sin_addr.s_addr = INADDR_ANY;
236     serv_addr.sin_port = htons(portno);
237
238     const int on = 1;
239     setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(
        on));
240     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(
        serv_addr)) < 0) {
241         perror("ERROR on binding");
242         exit(1);
243     }
244
245     /* Now start listening for the clients, here process will
246      * go in sleep mode and will wait for the incoming
          connection
247      */
248     listen(sockfd, 5);
249     clilen = sizeof(cli_addr);
250
251     pthread_t keyboard;
252     int worker_count = 0;
253     pthread_create(&keyboard, NULL, fun, (void*) &newsockfd);
254     /* If connection is established then start communicating
        */
```

```
255     while (1){
256        int worker_socket[max_conn];
257        int i;
258        printf("Waiting\n");
259        /* Accept actual connection from the client */
260        newsockfd = accept(sockfd, (struct sockaddr *) &
              cli_addr, &clilen);
261        printf("Connection %d\n", newsockfd-4);
262
263        socks[worker_count]=newsockfd;
264        if (newsockfd <= 0) {
265           perror("ERROR on accept");
266           break;
267        }
268        pthread_create(&(worker_thread[worker_count]),
269              NULL,
270              threads,
271              (void*) &newsockfd);
272        worker_count++;
273        /*for (i = 0; i < worker_count; i++) {
274              pthread_join(worker_thread[i], NULL);
275           }*/
276     }
277
278     return 0;
279 }
```

Файл сборки Makefile

```
1 all:
2     gcc main.c -o my_server -lm -lpthread
3
4 clean:
5     rm my_server
6     rm *.o
```

TCP клиент

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <netinet/in.h>
5 #include <ctype.h>
6 #define BUFLEN 40
7 int main(int argc, char *argv[])
```

```
8  {
9      int sockfd, portno, n;
10     struct sockaddr_in serv_addr;
11     struct hostent *server;
12
13     char buffer[BUFLEN];
14
15     if (argc < 3) {
16        fprintf(stderr,"usage␣%s␣hostname␣port\n", argv[0]);
17        exit(0);
18     }
19     portno = atoi(argv[2]);
20     /* Create a socket point */
21     sockfd = socket(AF_INET, SOCK_STREAM, 0);
22     if (sockfd < 0)
23     {
24        perror("ERROR␣opening␣socket");
25        exit(1);
26     }
27     server = gethostbyname(argv[1]);
28     if (server == NULL) {
29        fprintf(stderr,"ERROR,␣no␣such␣host\n");
30        exit(0);
31     }
32
33     bzero((char *) &serv_addr, sizeof(serv_addr));
34     serv_addr.sin_family = AF_INET;
35     /* bcopy((char *)server->h_addr,
36              (char *)&serv_addr.sin_addr.s_addr,
37                   server->h_length);*/
38     serv_addr.sin_port = htons(portno);
39
40     /* Now connect to the server */
41     if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
42     {
43        perror("ERROR␣connecting");
44        exit(1);
45     }
46     /* Now ask for a message from the user, this message
47      * will be read by server
48      */
49
50     while (1) {
51
52        printf("Input␣Expression:\n");
53        bzero(buffer, BUFLEN);
54        fgets(buffer, BUFLEN-1, stdin);
55        if (strcmp(buffer, "q\n") == 0) {
56           printf("Exit\n");
```

```
57            close(sockfd);
58            break;
59        }
60        n = write(sockfd, buffer, strlen(buffer));
61        if (n <= 0) {
62            printf("Connection lost!\n");
63            close(sockfd);
64            exit(0);
65        }
66
67        bzero(buffer, BUFLEN);
68        n = read(sockfd, buffer, BUFLEN-1);
69        if (n <= 0) {
70            printf("Connection lost!\n");
71            close(sockfd);
72            exit(0);
73        }
74
75        printf("%s\n", buffer);
76
77    }
78    return 0;
79 }
```

TCP клиент Windows

```
1
2
3  #include "stdafx.h"
4  #include <stdio.h>
5  #include <sys/types.h>
6  #include <winsock2.h>
7  #include <ws2tcpip.h>
8  #include <string.h>
9  #include <stdlib.h>
10 #include <ws2def.h>
11 #include <ctype.h>
12 #include "io.h"
13
14 #pragma comment (lib, "Ws2_32.lib")
15 #pragma comment (lib, "Mswsock.lib")
16 #pragma comment (lib, "AdvApi32.lib")
17
18 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
19 #define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void)
       0)
20
```

```
21 int main(int argc, _TCHAR* argv[])  //char *argv[]
22 {
23    //int portno, n;
24    int n;
25    int portno = 5001;
26    SOCKET sockfd;
27    WSADATA wsaData;
28    struct sockaddr_in serv_addr;
29    struct hostent *server;
30
31    char buffer[256];
32
33    /*if (argc < 3) {
34       fprintf(stderr,"usage %s hostname port\n", argv[0]);
35       exit(0);
36    }*/
37
38
39    //portno = atoi(argv[2]);
40    /* Create a socket point */
41    // Initialize Winsock
42
43     n = WSAStartup(MAKEWORD(2,2), &wsaData);
44     if (n != 0) {
45          printf("WSAStartup failed with error: %d\n", n);
46          return 1;
47     }
48    /* Create a socket point */
49     sockfd = socket(AF_INET, SOCK_STREAM, 0);
50     if (sockfd == INVALID_SOCKET)
51     {
52          perror("ERROR opening socket");
53          exit(1);
54     }
55
56    /*sockfd = socket(AF_INET, SOCK_STREAM, 0);
57    /*if (sockfd < 0)
58    {
59       perror("ERROR opening socket");
60       exit(1);
61    }*/
62    server = gethostbyname("192.168.56.101");
63    if (server == NULL) {
64         fprintf(stderr,"ERROR, no such host\n");
65         exit(0);
66     }
67    //server = gethostbyname(argv[1]);
68    /*if (server == NULL) {
69       fprintf(stderr,"ERROR, no such host\n");
```

```
70          exit(0);
71      }*/
72      bzero((char *) &serv_addr, sizeof(serv_addr));
73       serv_addr.sin_family = AF_INET;
74       bcopy((char *)server->h_addr,
75              (char *)&serv_addr.sin_addr.s_addr,
76                    server->h_length);
77       serv_addr.sin_port = htons(portno);
78
79      if (connect(sockfd,(sockaddr*) &serv_addr,sizeof(serv_addr
           )) <0)
80      {
81          perror("ERROR connecting");
82          exit(1);
83      }
84      while (1) {
85          printf("Input Expression:\n");
86          bzero(buffer, 256);
87          fgets(buffer, 255, stdin);
88          if (strcmp(buffer, "q\n") == 0) {
89              printf("Exit\n");
90              closesocket(sockfd);
91              break;
92          }
93          n = send(sockfd, buffer, strlen(buffer),0);
94          if (n < 0) {
95              perror("ERROR writing to socket");
96              exit(1);
97          }
98          bzero(buffer, 256);
99          n = recv(sockfd, buffer, 255,0);
100         if (n < 0) {
101             perror("ERROR reading from socket");
102             exit(1);
103         }
104         printf("%s\n", buffer);
105     }
106     return 0;
107 }
```

UDP сервер Windows

```
1
2 #define _CRT_SECURE_NO_DEPRECATE
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <winsock2.h>
```

```
6  #include <ws2tcpip.h>
7  #include <string.h>
8  #include <assert.h>
9  #include <math.h>
10 #include <conio.h>
11
12 #pragma comment (lib, "Ws2_32.lib")
13 #pragma comment (lib, "Mswsock.lib")
14 #pragma comment (lib, "AdvApi32.lib")
15
16 #define BUFLEN 40  //Max length of buffer
17 #define PORT 5001   //The port on which to listen for
       incoming data
18 #define snprintf _snprintf
19
20 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
21 #define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void)
       0)
22
23 struct connection {
24    int recv;
25    char data[255];
26    struct sockaddr_in addr;
27    int socket, addrlen;
28 };
29
30 struct connection conns[20];
31 int conn_num = 0;
32 DWORD dwThreadId[5];
33
34 void my_read(int conn, char* data, int length) {
35    while (!conns[conn].recv) Sleep(1);
36    bcopy(conns[conn].data, data, length);
37    conns[conn].recv = 0;
38 }
39
40 int my_write(int conn, char* data, int length) {
41    int n;
42    printf("%d\n", conns[conn].socket);
43    n=sendto(conns[conn].socket, data, length, 0,
44       (struct sockaddr*) &conns[conn].addr, conns[conn].
           addrlen);
45    if(n==SOCKET_ERROR)
46       printf("Error\n");
47    printf("%d␣\n",n);
48    return n;
49 }
50
51 long int factorial(long int x) {
```

```
52     return !x ? 1 : x * factorial(x - 1);
53 }
54 void calculation(char *buffer,char *answer) {
55     int i, j;
56     long int f;
57     double a, b, k;
58     char temp1[BUFLEN],temp2[BUFLEN];
59     int size;
60     printf("Received: %s\n", buffer);
61     bzero(temp1, BUFLEN);
62     bzero(temp2, BUFLEN);
63     for (i = 1; i < 255; i++) {
64         //factorial
65         if (buffer[i] == '!'){
66             strncpy(temp1, buffer, i);
67             f = atoi(temp1);
68             printf("f = %d\n", f);
69             break;
70         }
71         //sqrt
72         if (buffer[i] == '#'){
73             strncpy(temp1, buffer, i);
74             a = atof(temp1);
75             printf("k = %f\n", a);
76             break;
77         }
78         // *, /, +,-
79         if ((buffer[i] == '*') ||(buffer[i] == '/') ||(buffer[i
               ] == '+') || (buffer[i] == '-'))  {
80             strncpy(temp1, buffer, i);
81             a = atof(temp1);
82             printf("a = %f\n", a);
83             break;
84         }
85     }
86     //end expression
87     for (j = 0; j < BUFLEN-1; j++) {
88         if (buffer[j] == '=') {
89             strncpy(temp2, buffer + i + 1, j - i - 1);
90             b = atof(temp2);
91             printf("b = %f\n", b);
92             break;
93         }
94     }
95     if (buffer[i] == '+'){
96         Sleep(10000);
97         sprintf(answer, "Answer = %f", a+b);
98     }
99     if (buffer[i] == '-'){
```

```
100         sprintf(answer, "Answer = %f", a-b);
101     }
102     if (buffer[i] == '*'){
103         sprintf(answer, "Answer = %f", a*b);
104     }
105     if (buffer[i] == '/'){
106         if (b==0){
107             sprintf(answer, "Error, input divider != 0 !\n");
108         }
109         else {
110             sprintf(answer, "Answer = %f", a/b);
111         }
112     }
113     if (buffer[i] == '#'){
114         if (a<0){
115             sprintf(answer, "Error, input Root >= 0 !\n");
116         }
117         else {
118             sprintf(answer, "Sqrt = %f", sqrt(a));
119         }
120     }
121     if (buffer[i] == '!'){
122         if (f<=0){
123             sprintf(answer, "Error, input Factorial > 0 !\n");
124         }
125         else {
126             sprintf(answer, "Factorial = %d", factorial(f));
127         }
128     }
129 }
130 
131 void die(char *s)
132 {
133     perror(s);
134     exit(1);
135 }
136 
137 DWORD WINAPI startThread(LPVOID lpParam) {
138     int sock = *(int*)lpParam;
139     char mesg[BUFLEN];
140     while(1)
141     {
142         char answer[BUFLEN];
143         bzero(mesg, sizeof(mesg));
144         my_read(sock, mesg, BUFLEN);
145         calculation(mesg,answer);
146         my_write(sock, answer, strlen(answer));
147         printf("%s\n",answer);
148     }
```

```
149 }
150
151 int main() {
152    //struct mySocket s;
153    int n;
154    int num=0; //количество циклов для shutdown
155    struct sockaddr_in servaddr, cliaddr,portno;
156    int sock;
157    socklen_t len;
158    len = sizeof(cliaddr);
159    char mesg[BUFLEN];
160    HANDLE thread[5],mainthread;
161
162    // Initialize Winsock
163    WSADATA wsaData;
164    n = WSAStartup(MAKEWORD(2,2), &wsaData);
165    if (n != 0) {
166       printf("WSAStartup␣failed␣with␣error:␣%d\n", n);
167       return -1;
168    }
169
170    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
171    if (sock == INVALID_SOCKET)
172    {
173       die("socket");
174    }
175
176    memset((char *) &servaddr, 0, sizeof(servaddr));
177    bzero(&servaddr, sizeof(servaddr));
178    servaddr.sin_family = AF_INET;
179    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
180    servaddr.sin_port = htons(PORT);
181
182    if( bind(sock , (struct sockaddr*)&servaddr, sizeof(
          servaddr) ) == -1)
183    {
184       die("bind");
185    }
186    while(1)
187    {
188
189       /* Accept actual connection from the client */
190
191       /* If connection is established then start
             communicating */
192       int n, i, num;
193       char buff[256];
194       bzero(buff, 255);
195
```

```
196         struct sockaddr_in client_addr;
197         socklen_t addrlen = sizeof(struct sockaddr_in);
198         n = recvfrom(sock, buff, 255, 0, (struct sockaddr*) &
                client_addr, &addrlen);
199         printf("Res␣%s␣\n",buff);
200         num = -1;
201         for (i = 0; i < conn_num; i++) {
202             if ((conns[i].addr.sin_addr.s_addr == client_addr.
                    sin_addr.s_addr) &&
203                 (conns[i].addr.sin_port == client_addr.sin_port))
                        {
204                     num = i;
205                     break;
206             }
207         }
208         if (num == -1) {
209             num = conn_num;
210             conns[num].addr.sin_addr = client_addr.sin_addr;
211             conns[num].addr.sin_port = client_addr.sin_port;
212             conns[num].addr.sin_family = client_addr.sin_family;
213             //conns[num].addr.sin_zero = client_addr.sin_zero;
214             conns[num].socket = sock;
215             conns[num].addrlen = addrlen;
216             conns[num].recv = 0;
217             thread[num]= CreateThread(NULL, 0, startThread, (
                    LPVOID)&num, 0, &dwThreadId[i]);
218             conn_num++;
219         }
220         printf("Client␣%d\n", num);
221 
222         bcopy(buff, conns[num].data, 255);
223         conns[num].recv = 1;
224     }
225     closesocket(sock);
226     return 0;
227 
228 }
```

UDP клиент

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <string.h>
7 
```

```c
#define BUFLEN 40

int main(int argc, char*argv[])
{
    int sockfd,n,portno;
    struct sockaddr_in servaddr,cliaddr;
    char sendline[BUFLEN];
    char recvline[BUFLEN];

    portno = atoi(argv[2]);
    sockfd=socket(AF_INET,SOCK_DGRAM,0);

    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr=inet_addr(argv[1]);
    servaddr.sin_port=htons(portno);

    //Start connect
    while (1) {
        printf("\nInput␣Expression:\n");
        if(fgets(sendline, BUFLEN,stdin) != NULL)
            sendto(sockfd,sendline,strlen(sendline),0,
                    (struct sockaddr *)&servaddr,sizeof(
                        servaddr));
        /*if (sendline[strlen(sendline)-2]== '!' || sendline[
            strlen(sendline)-2]== '#') {
                    printf("Wait...\n");
                }*/
        if (strcmp(sendline, "q\n") == 0) {
                    printf("Exit\n");
                    close(sockfd);
                    break;
        }
        bzero(recvline,BUFLEN);
        n=recvfrom(sockfd,recvline,BUFLEN,0,NULL,NULL);
        printf("%s\n", recvline);
    }
    return 0;
}
```