

# Сети ЭВМ и телекоммуникации

Д. С. Мурашко

29 декабря 2014 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 минусом приложения. Решить эту проблему возможно с помощью вычислений с произвольной точностью, используя, к примеру, библиотеку GMP.
2. Ограничения на длину сообщения: максимальный размер 40 символов. Для правильного чтения/отправки сообщения требуется фиксированная длина.
3. Ограничение, 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 ТСР

### 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. Предусмотрены 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:

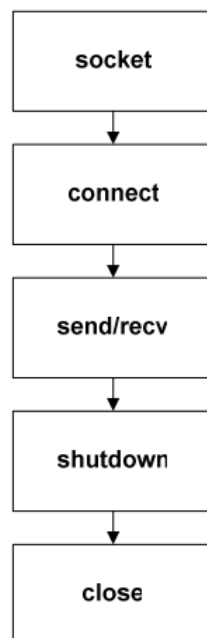
(a) Число

(b) Знак операции (!)

## 2.2 Архитектура приложения

### 2.2.1 Описание клиента

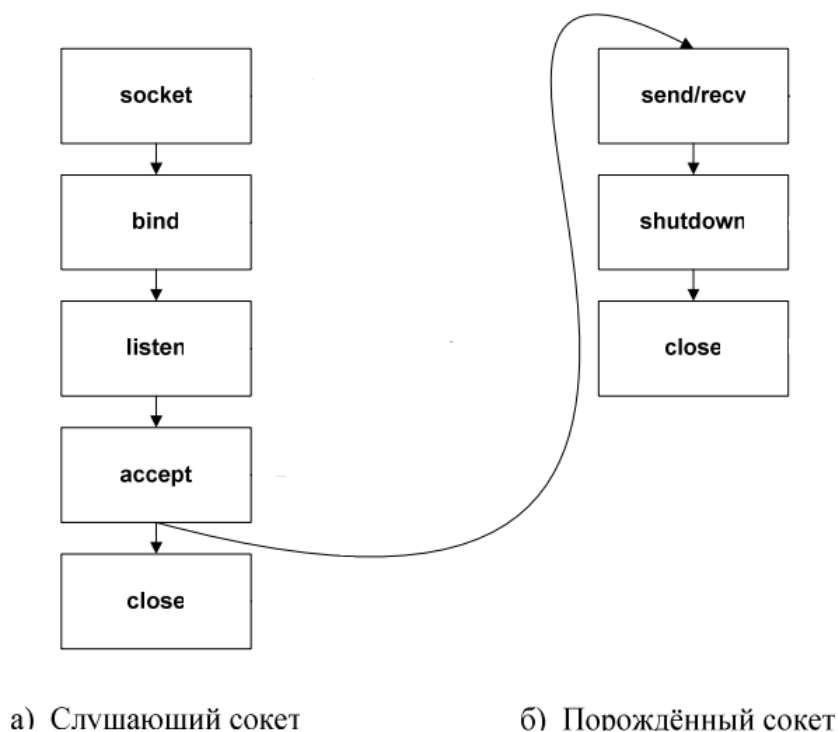
Клиент протокола ТСР создаёт экземпляр сокета, необходимый для взаимодействия с сервером, организует соединение, осуществляет обмен данными, в соответствии с протоколом прикладного уровня. Структура ТСР-клиента представлена ниже:



### 2.2.2 Описание сервера

Организация ТСР-сервера отличается от ТСР-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 ТСП-сервера и взаимосвязь сокетов изображена ниже:



### 2.2.3 Дизайн приложения

После запуска сервер начинает прослушивать определенный порт (5001) и при подключении клиента, выделяет ему отдельный поток. При подключении 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 предлагается ввести следующее выражение.

Результат вычисления "долгих" математических операций (факториал, квадратный корень) посылаются с отложенной посылкой результата клиенту. Для получения результата клиент вводит команду СHECK-F или СHECK-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

Протестируем вначале быстрые операции: сложение, вычитание, 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 приложения с слишком большими числами:

Input Expression:

34567890456789456784567+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, единственное различие - реализация многопоточности. Так как в данной реализации не предусматривается установления логического соединения, то удобна организация асинхронного сервера, реагирующего на сообщения. Поэтому реализация такой задачи была осуществлена с помощью логики, а не потоков.

# 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/resv`, т.к. он может считать лишь часть сообщения. Таким образом, для чтения одного сообщения может понадобиться несколько вызовов `read/resv`. Поэтому в данном приложении длина сообщения ограничена. 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

### ТСР 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(
137                 "Too_long_operand_1!\n"));
138         }else
139             a = atof(temp1);
140             break;
141     }
142     // *, /, +, -
143     if ((buffer[i] == '*' || (buffer[i] == '/') || (buffer[i]
144         ] == '+') || (buffer[i] == '-')) {
145         strncpy(temp1, buffer, i);
146         if (strlen(temp1)>20){
147             write(newsockfd, "Too_long_operand_1!\n", strlen(
148                 "Too_long_operand_1!\n"));
149         }else
150             a = atof(temp1);
151             break;
152     }
153 }
154 //end expression
155 for (j = 0; j < (BUFLen-1); j++) {
156     if (buffer[j] == '=') {
157         strncpy(temp2, buffer + i + 1, j - i - 1);
158         if (strlen(temp2)>20){
159             write(newsockfd, "Too_long_operand_2!\n", strlen(
160                 "Too_long_operand_2!\n"));
161         }else
162             b = atof(temp2);
163             break;
164     }
165 }
166
167 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",
182             strlen("Error, input_divider!=0!\n"));
183     }
184     else {
185         answer = a/b;
186         sprintf(ans, "Answer_=%lf\n", answer);
187         write(newsockfd, ans, strlen(ans));
188     }
189 }
190 if (buffer[i] == '#') {
191     if (a<0) {
192         write(newsockfd, "Error, input_Root>=0!\n", strlen(
193             "Error, input_Root>=0!\n"));
194     }
195     else {
196         sqrt_args[newsockfd] = a;
197         sqrt_res[newsockfd] = 0;
198         pthread_create(&(sqrt_thread[newsockfd]),
199             NULL,
200             run_sqrt,
201             (void*) &newsockfd);
202         write(newsockfd, "Wait\n", strlen("Wait\n"));
203     }
204 }
205 if (buffer[i] == '!') {
206     if (f<=0) {
207         write(newsockfd, "Error, input_Factorial>=0!\n",
208             strlen("Error, input_Factorial>=0!\n"));
209     }
210     else {
211         fact_args[newsockfd] = f;
212         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
247        connection
248        */
249     listen(sockfd, 5);
250     clilen = sizeof(cli_addr);
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, BUFLLEN);
68     n = read(sockfd, buffer, BUFLLEN-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)
20     0)

```

```

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 = WSASStartup(MAKEWORD(2,2), &wsaData);
44 |     if (n != 0) {
45 |         printf("WSASStartup 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)
80 )) <0)
81 {
82     perror("ERROR_␣connecting");
83     exit(1);
84 }
85 while (1) {
86     printf("Input_␣Expression:\n");
87     bzero(buffer, 256);
88     fgets(buffer, 255, stdin);
89     if (strcmp(buffer, "q\n") == 0) {
90         printf("Exit\n");
91         closesocket(sockfd);
92         break;
93     }
94     n = send(sockfd, buffer, strlen(buffer),0);
95     if (n < 0) {
96         perror("ERROR_␣writing_␣to_␣socket");
97         exit(1);
98     }
99     bzero(buffer, 256);
100    n = recv(sockfd, buffer, 255,0);
101    if (n < 0) {
102        perror("ERROR_␣reading_␣from_␣socket");
103        exit(1);
104    }
105    printf("%s\n", buffer);
106 }
107 return 0;
108 }

```

## UDP сервер Windows

```

1
2 #define _CRT_SECURE_NO_DEPRECATED
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
11 #pragma comment (lib, "Ws2_32.lib")
12 #pragma comment (lib, "Mswsock.lib")
13 #pragma comment (lib, "AdvApi32.lib")
14
15 #define BUFLen 100 //Max length of buffer
16 #define PORT 5001 //The port on which to listen for
    incoming data
17 #define snprintf _snprintf
18
19 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
20 #define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void)
    0)
21
22
23 long int factorial(long int x) {
24     return !x ? 1 : x * factorial(x - 1);
25 }
26 void calculation(char *buffer, char *answer) {
27     int i, j;
28     long int f;
29     double a, b, k;
30     char temp1[BUFLen], temp2[BUFLen];
31     int size;
32     printf("Received: \u%s\n", buffer);
33     bzero(temp1, BUFLen);
34     bzero(temp2, BUFLen);
35     for (i = 1; i < 255; i++) {
36         //factorial
37         if (buffer[i] == '!'){
38             strncpy(temp1, buffer, i);
39             f = atoi(temp1);
40             printf("f\u=\u%d\n", f);
41             break;
42         }
43         //sqrt
44         if (buffer[i] == '#'){
45             strncpy(temp1, buffer, i);
46             a = atof(temp1);
47             printf("k\u=\u%f\n", a);
48             break;
49         }
50         // *, /, +, -
51         if ((buffer[i] == '*' || (buffer[i] == '/' || (buffer[i]
            ] == '+' || (buffer[i] == '-')) {

```



```

52         strncpy(temp1, buffer, i);
53         a = atof(temp1);
54         printf("a_=%f\n", a);
55         break;
56     }
57 }
58 //end expression
59 for (j = 0; j < BUFLen-1; j++) {
60     if (buffer[j] == '=') {
61         strncpy(temp2, buffer + i + 1, j - i - 1);
62         b = atof(temp2);
63         printf("b_=%f\n", b);
64         break;
65     }
66 }
67 if (buffer[i] == '+'){
68     sprintf(answer, "Answer_=%f", a+b);
69 }
70 if (buffer[i] == '-'){
71     sprintf(answer, "Answer_=%f", a-b);
72 }
73 if (buffer[i] == '*'){
74     sprintf(answer, "Answer_=%f", a*b);
75 }
76 if (buffer[i] == '/'){
77     if (b==0){
78         sprintf(answer, "Error,input_divider!=0!\n");
79     }
80     else {
81         sprintf(answer, "Answer_=%f", a/b);
82     }
83 }
84 if (buffer[i] == '#'){
85     if (a<0){
86         sprintf(answer, "Error,input_Root>=0!\n");
87     }
88     else {
89         sprintf(answer, "Sqrt_=%f", sqrt(a));
90     }
91 }
92 if (buffer[i] == '!'){
93     if (f<=0){
94         sprintf(answer, "Error,input_Factorial>0!\n");
95     }
96     else {
97         sprintf(answer, "Factorial_=%d", factorial(f));
98     }
99 }
100 }

```

```

101
102 void die(char *s)
103 {
104     perror(s);
105     exit(1);
106 }
107
108 int main() {
109     //struct mySocket s;
110     int n;
111     int num=0; //количество циклов для shutdown
112     struct sockaddr_in servaddr, cliaddr, portno;
113     int sock;
114     socklen_t len;
115     len = sizeof(cliaddr);
116     char mesg[BUFLEN];
117
118     // Initialize Winsock
119     WSADATA wsaData;
120     n = WSASStartup(MAKEWORD(2,2), &wsaData);
121     if (n != 0) {
122         printf("WSAStartup failed with error: %d\n", n);
123         return -1;
124     }
125
126     sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
127     if (sock == INVALID_SOCKET)
128     {
129         die("socket");
130     }
131
132     memset((char *) &servaddr, 0, sizeof(servaddr));
133     bzero(&servaddr, sizeof(servaddr));
134     servaddr.sin_family = AF_INET;
135     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
136     servaddr.sin_port = htons(PORT);
137
138     if( bind(sock , (struct sockaddr*)&servaddr, sizeof(
servaddr) ) == -1)
139     {
140         die("bind");
141     }
142     while(1)
143     {
144         char answer[BUFLEN];
145         bzero(mesg, sizeof(mesg));
146         recvfrom(sock, mesg, BUFLEN, 0, (struct sockaddr
*) &cliaddr, &len);
147         calculation(mesg, answer);

```

```

148         sendto(sock, answer, strlen(answer), 0, (struct
            sockaddr *)&cliaddr, sizeof(cliaddr));
149         printf("%s\n", answer);
150     }
151     closesocket(sock);
152     return 0;
153 }
154 }

```

## 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
8  #define BUFLen 100
9
10 int main(int argc, char*argv[])
11 {
12     int sockfd,n,portno;
13     struct sockaddr_in servaddr,cliaddr;
14     char sendline[BUFLen];
15     char recvline[BUFLen];
16
17     portno = atoi(argv[2]);
18     sockfd=socket(AF_INET,SOCK_DGRAM,0);
19
20     bzero(&servaddr, sizeof(servaddr));
21     servaddr.sin_family = AF_INET;
22     servaddr.sin_addr.s_addr=inet_addr(argv[1]);
23     servaddr.sin_port=htons(portno);
24
25     //Start connect
26     while (1) {
27         printf("\nInput Expression:\n");
28         if(fgets(sendline, BUFLen,stdin) != NULL)
29             sendto(sockfd,sendline,strlen(sendline),0,
30                 (struct sockaddr *)&servaddr, sizeof(
                    servaddr));
31         /*if (sendline[strlen(sendline)-2]== '!' || sendline[
            strlen(sendline)-2]== '#') {
32             printf("Wait...\n");
33         }*/
34         if (strcmp(sendline, "q\n") == 0) {

```

```
35         printf("Exit\n");
36         close(sockfd);
37         break;
38     }
39     bzero(recvline,BUFLEN);
40     n=recvfrom(sockfd,recvline,BUFLEN,0,NULL,NULL);
41     printf("%s\n", recvline);
42 }
43 return 0;
44 }
```