**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Лабораторная работа № 6

| | |
|---|---|
| **Дисциплина** | Операционные системы. |
| **Тема** | Сокеты. |
| | |
| **Студент** | Сиденко А.Г. |
| **Группа** | ИУ7-63Б |
| **Оценка (баллы)** | |
| **Преподаватель** | Рязанова Н.Ю. |

Москва, 2020 г.

**Задание 1:** Написать приложение по модели клиент-сервер, демонстрирующее взаимодействие параллельных процессов на отдельном компьютере с использованием сокетов в файловом пространстве имен: семейство - AF_UNIX, тип - SOCK_DGRAM. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

Листинг 1: Код сервера

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCK_NAME "socket.soc"
#define MSG_SIZE 256

int main()
{
  // Для сокетов Unix (сокетов в файловом пространстве имен)
  // есть специализированная структура sockaddr_un
  struct sockaddr_un server;
  char msg[MSG_SIZE];
  int bytes;

  // Создание сокета в файловом пространстве имен (домен AF_UNIX)
  // Тип сокета -- SOCK_DGRAM означает датаграммный сокет
  // Протокол -- 0, протокол выбирается по умолчанию
  int sock = socket(AF_UNIX, SOCK_DGRAM, 0);
  if (sock < 0)
  {
    printf("%s", strerror(errno));
    return errno;
  }

  // Укажем семейство адресов, которыми мы будем пользоваться
  server.sun_family = AF_UNIX;
  // Укажем имя файла сокета
  strcpy(server.sun_path, SOCK_NAME);

  // Связывание сокета с заданным адресом
  // bind(дескриптор сокета, указатель на структуру, длина структуры)
  if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
  {
    printf("%s", strerror(errno));
    return errno;
  }

```

```
43    // Наша программа-сервер становится доступна для соединения
44    // по заданному адресу (имени файла)
45
46    // Пока клиент не отправит сообщение "break"
47    while (strcmp(msg, "break"))
48    {
49      // Для чтения данных из датаграммного сокета - recvfrom,
50      // которая блокирует программу до тех пор, пока на входе не появятся
51      // новые данные
52      // Так как нас не интересуют данные об адресе клиента
53      // передаем значения NULL в предпоследнем и последнем параметрах
54      bytes = recvfrom(sock, msg, MSG_SIZE, 0, NULL, NULL);
55      if (bytes < 0)
56      {
57        printf("%s", strerror(errno));
58        return errno;
59      }
60      // Символ окончания строки
61      msg[bytes] = 0;
62      printf("Сообщение от клиента: %s\n", msg);
63    }
64
65    // Закрываем сокет
66    close(sock);
67    // Удаляем файл сокета
68    unlink(SOCK_NAME);
69
70    return errno;
71  }
```

Листинг 2: Код клиента

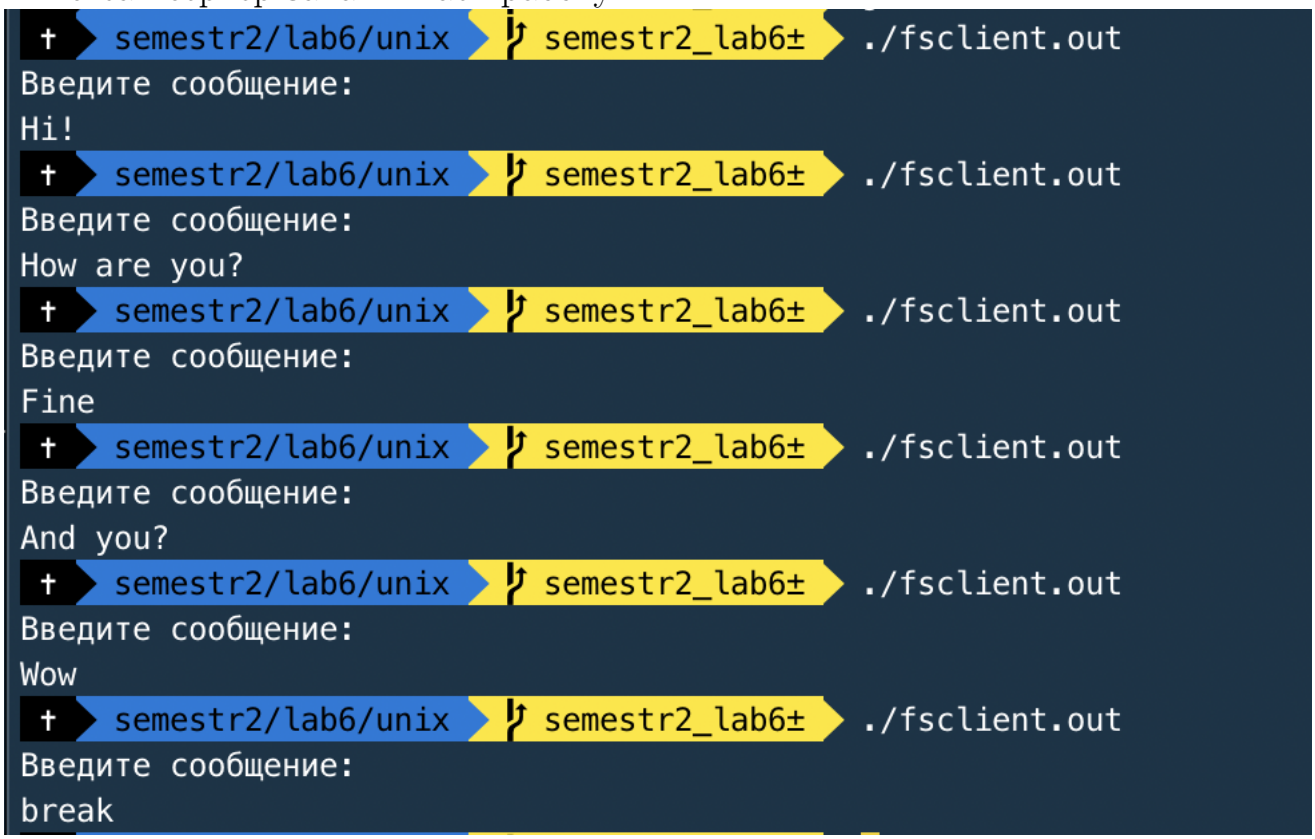
```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <errno.h>
5  #include <sys/socket.h>
6  #include <sys/un.h>
7
8  #define SOCK_NAME "socket.soc"
9  #define MSG_SIZE 256
10
11 int main(int argc, char ** argv)
12 {
13   // Создание сокета в файловом пространстве имен (домен AF_UNIX)
14   // Тип сокета -- SOCK_DGRAM означает датаграммный сокет
15   // Протокол -- 0, протокол выбирается по умолчанию
16   char msg[MSG_SIZE];
17   struct sockaddr_un server;
18
19   int sock = socket(AF_UNIX, SOCK_DGRAM, 0);
20   if (sock < 0)
```

```
21      {
22          printf("%s", strerror(errno));
23          return errno;
24      }
25
26      // Укажем семейство адресов, которыми мы будем пользоваться
27      server.sun_family = AF_UNIX;
28      // Укажем имя файла сокета
29      strcpy(server.sun_path, SOCK_NAME);
30
31      // Приглашение и ввод сообщения для сервера
32      printf("Введите сообщение:\n");
33      scanf("%s", msg);
34
35      // Передаем сообщение серверу
36      // sendto(дескриптор сокета, адрес буфера для передачи данных,
37      // его длина, дополнительные флаги, адрес сервера, его длине)
38      sendto(sock, msg, strlen(msg), 0, (struct sockaddr *) &server,
39                                                          sizeof(server));
40
41      return errno;
42  }
```

Запускаем приложение сервера и принимаем сообщения от клиентов:

```
t  semestr2/lab6/unix  semestr2_lab6±  ./fsserver.out
Сообщение от клиента: Hi!
Сообщение от клиента: How
Сообщение от клиента: Fine
Сообщение от клиента: And
Сообщение от клиента: Wow
Сообщение от клиента: break
```

Запускаем приложения клиентов, отправляем сообщения, при отправке сообщения break сервер заканчивает работу:



**Задание 2:** Написать приложение по модели клиент-сервер, осуществляющее взаимодействие параллельных процессов, которые выполняются на разных компьютерах. Для взаимодействия с клиентами сервер должен использовать мультиплексирование. Сервер должен обслуживать запросы параллельно запущенных клиентов. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

Листинг 3: Код сервера

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <errno.h>
4  #include <strings.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <netinet/in.h>
8  #include <unistd.h>
9
10 #define MSG_SIZE 256
11 #define LISTENQ 1024
12
13 int maxi, maxfd;
14
15 // Соединение с новым клиентом
16 int newClient(int listensock, int client[FD_SETSIZE], fd_set *allset,
17                                                       fd_set *reset)
18 {
19   // Индекс
20   int i;
21   int connfd;
22
23   if (FD_ISSET(listensock, reset))
24   {
25     // Установка соединения в ответ на запрос клиента
26     // Так как нас не интересуют данные об адресе клиента
27     // передаем значения NULL в предпоследнем и последнем параметрах
28     int connfd = accept(listensock, NULL, NULL);
29     if (connfd < 0)
30     {
31       printf("%s", strerror(errno));
32       return errno;
33     }
34     // Функция accept() возвращает новый сокет, открытый
35     // для обмена данными с клиентом, запросившим соединение
36
37     // Сохраняем дескриптор в первый свободный
38     for (i = 0; i < FD_SETSIZE; i++)
39     {
40       if (client[i] < 0)
41       {
42         client[i] = connfd;
```

```
43            break;
44          }
45        }
46
47        if (i == FD_SETSIZE)
48        {
49          printf("Достигнуто максимальное число клиентов");
50          return errno;
51        }
52
53        // Добавление нового дескриптора
54        FD_SET(connfd, allset);
55
56        // Максимальный для функции select
57        if (connfd > maxfd)
58          maxfd = connfd;
59
60        // Максимальный индекс в массиве клиентов
61        if (i > maxi)
62          maxi = i;
63        printf("Клиент %d подключился\n", i);
64    }
65    return errno;
66  }
67
68  int readMsg(int client[FD_SETSIZE], fd_set *allset, fd_set *reset)
69  {
70    int n, i;
71    int sockfd;
72    char msg[MSG_SIZE];
73    // Проверяем все клиенты на наличие данных, пока не дошли до конца
74    // или не закончились дескрипторы готовые для чтения
75    for (i = 0; i <= maxi; i++)
76    {
77      // Если не пустой
78      if ((sockfd = client[i]) > 0)
79      {
80        // Установлен ли бит?
81        if (FD_ISSET(sockfd, reset))
82        {
83          // Соединение закрыто клиентом
84          if ((n = read(sockfd, msg, MSG_SIZE)) == 0)
85          {
86            // Закрываем сокет
87            close(sockfd);
88            // Сброс бита
89            FD_CLR(sockfd, allset);
90            // Освобождаем ячейку в массиве клиентов
91            client[i] = -1;
92            printf("Клиент %d отключился\n", i);
93          }
94          else
```

```
95          {
96            // Сообщение клиенту о доставке сообщения
97            write(sockfd, "OK", 2);
98            // Установка символа конца строки и вывод сообщения на экран
99            msg[n] = 0;
100           printf("Сообщение от клиента %d: %s", i, msg);
101         }
102       }
103     }
104   }
105   return errno;
106 }
107
108 int main(int argc, char ** argv)
109 {
110   int listensock;
111   int client[FD_SETSIZE];
112   fd_set reset, allset;
113   // Структура предназначен для хранения адресов в формате Интернета
114   struct sockaddr_in server;
115
116   if (argc < 2)
117   {
118     fprintf(stderr, "Использование: %s <port_number>\n", argv[0]);
119     return 1;
120   }
121
122   // Создание сетевого сокета (домен AF_INET)
123   // Тип сокета — SOCK_STREAM, сокет должен быть потоковым
124   // Протокол — 0, протокол выбирается по умолчанию
125   listensock = socket(AF_INET, SOCK_STREAM, 0);
126   if (listensock < 0)
127   {
128     printf("%s", strerror(errno));
129     return errno;
130   }
131
132   // Укажем семейство адресов, которыми мы будем пользоваться
133   server.sin_family = AF_INET;
134   // Укажем адрес (наша программа-сервер зарегистрируется на всех адресах
135   // машины, на которой она выполняется)
136   server.sin_addr.s_addr = INADDR_ANY;
137   // Укажем значение порта. Функция htons() переписывает
138   // двухбайтовое значение порта так, чтобы порядок байтов
139   // соответствовал принятому в Интернете
140   server.sin_port = htons(atoi(argv[1]));
141
142   // Связывание сокета с заданным адресом
143   // bind(дескриптор сокета, указатель на структуру, длина структуры)
144   if (bind(listensock, (struct sockaddr *) &server, sizeof(server)) < 0)
145   {
146     printf("%s", strerror(errno));
```

```
147       return errno;
148     }
149
150     // Переводим сервер в режим ожидания запроса на соединение
151     // Второй параметр - максимальное число обрабатываемых
152     // одновременно соединений
153     listen(listensock, LISTENQ);
154
155     // Инициализация значения
156     maxfd = listensock;
157     // Индекс в массиве клиентов (наибольший используемый)
158     maxi = -1;
159     // Массив дескрипторов присоединенного сокета для каждого клиента
160     for (int i = 0; i < FD_SETSIZE; i++)
161       client[i] = -1; // -1 означает, что элемент свободен
162
163     // Сбрасываем все биты в allset
164     FD_ZERO(&allset);
165     // Устанавливаем бит для listensock в allset
166     FD_SET(listensock, &allset);
167
168     while(1)
169     {
170       // Присваивание значения структуре
171       reset = allset;
172       // select() ждет пока не будет установлено новое клиентское
173       // соединение или на существующем не прибудут данные
174       // select(количество проверяемых дескрипторов, 2-4 наборы
175       // дескрипторов, которые следует проверять, интервал времени,
176       // по прошествии которого она вернет управление в любом случае)
177       select(maxfd + 1, &reset, NULL, NULL, NULL);
178
179       if (newClient(listensock, client, &allset, &reset) ||
180                                           readMsg(client, &allset, &reset))
181         return errno;
182     }
183
184     // Закрываем сокет
185     close(listensock);
186
187     return errno;
188 }
```

Листинг 4: Код клиента

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

```
 8 #include <netdb.h>
 9 #include <unistd.h>
10
11 #define MSG_SIZE 256
12
13 int main(int argc, char ** argv)
14 {
15   struct sockaddr_in server;
16   struct hostent *host;
17   char msg_client[MSG_SIZE], msg_server[MSG_SIZE];
18
19   if (argc < 3)
20   {
21     fprintf(stderr, "Использование: %s <hostname> <port_number>\n",
22                                                         argv[0]);
23     return 1;
24   }
25
26   // Создание сетевого сокета (домен AF_INET)
27   // Тип сокета — SOCK_STREAM, сокет должен быть потоковым
28   // Протокол — 0, протокол выбирается по умолчанию
29   int sock = socket(AF_INET, SOCK_STREAM, 0);
30   if (sock < 0)
31   {
32     printf("%s", strerror(errno));
33     return errno;
34   }
35
36   // Преобразование доменного имени сервера в его сетевой адрес
37   host = gethostbyname(argv[1]);
38   if (host == NULL)
39   {
40     printf("%s", strerror(errno));
41     return errno;
42   }
43
44   // Укажем семейство адресов, которыми мы будем пользоваться
45   server.sin_family = AF_INET;
46   // Укажем адрес (наша программа-сервер зарегистрируется
47   // на всех адресах машины, на которой она выполняется)
48   memcpy(&server.sin_addr, host->h_addr_list[0], host->h_length);
49   // Укажем значение порта. Функция htons() переписывает
50   // двухбайтовое значение порта так, чтобы порядок байтов
51   // соответствовал принятому в Интернете
52   server.sin_port = htons(atoi(argv[2]));
53
54   // Установка соединения
55   if (connect(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
56   {
57     printf("%s", strerror(errno));
58     return errno;
59   }
```

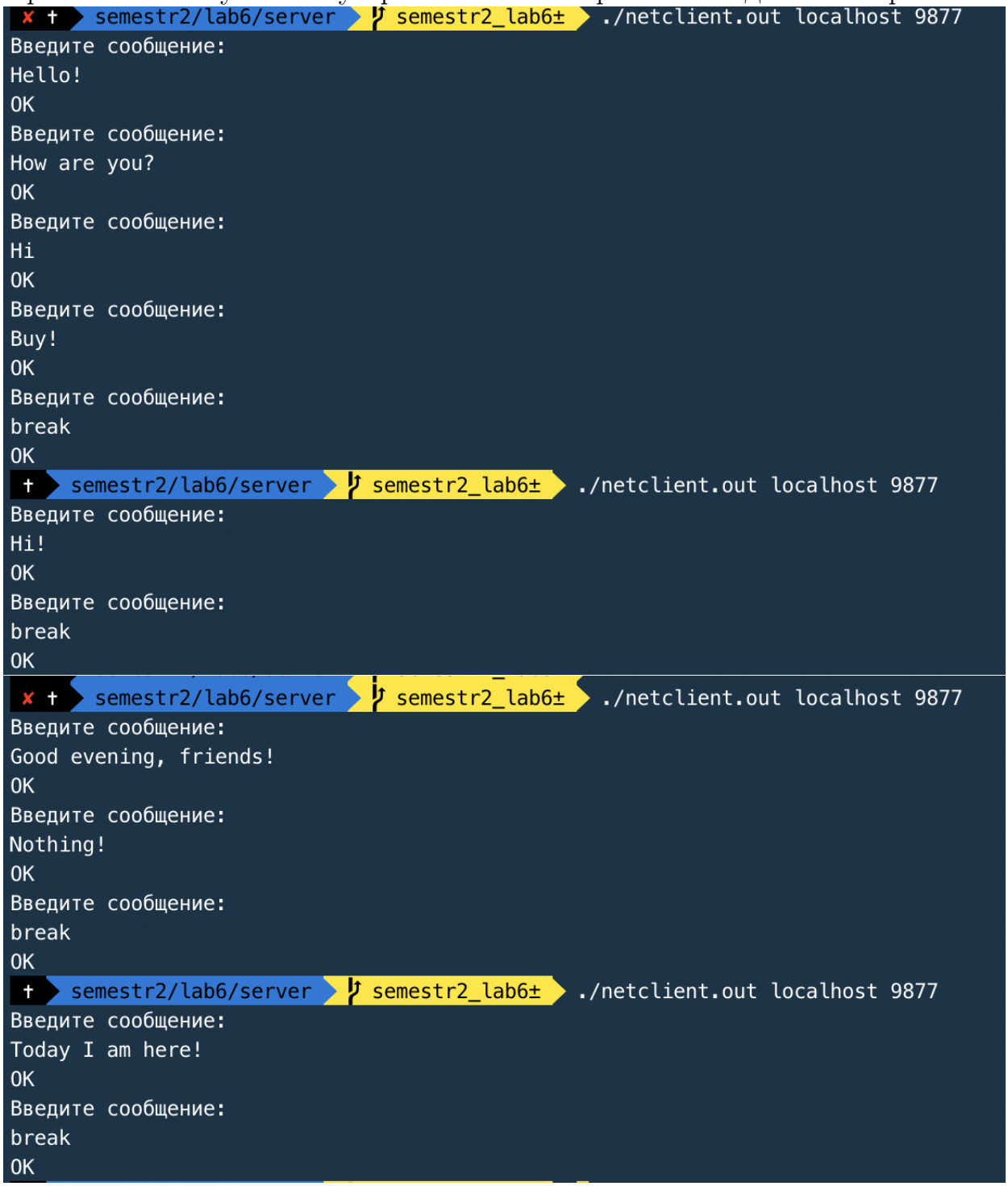
```
60
61      // Пока не сообщение "break"
62      while (strcmp(msg_client, "break\n"))
63      {
64          memset(msg_client, 0, MSG_SIZE);
65          printf("Введите сообщение:\n");
66          fgets(msg_client, MSG_SIZE, stdin);
67          // Для записи данных - write
68          write(sock, msg_client, strlen(msg_client));
69
70          // Заполнение массива нулями
71          memset(msg_server, 0, MSG_SIZE);
72          // Для чтения данных из сокета - read
73          read(sock, msg_server, MSG_SIZE);
74          printf("%s\n", msg_server);
75      }
76
77      // Закрываем сокет
78      close(sock);
79
80      return 0;
81  }
```

Запускаем приложение сервера и принимаем сообщения от клиентов:

```
✘ † semestr2/lab6/server  semestr2_lab6± ./netserver.out 9877
Клиент 0 подключился
Клиент 1 подключился
Сообщение от клиента 0: Hi!
Сообщение от клиента 1: Hello!
Сообщение от клиента 1: How are you?
Клиент 2 подключился
Сообщение от клиента 0: I am fine. And you?
Сообщение от клиента 2: Good evening, friends!
Сообщение от клиента 1: Hi
Сообщение от клиента 0: Hello, client!
Сообщение от клиента 1: Buy!
Сообщение от клиента 1: break
Клиент 1 отключился
Клиент 1 подключился
Сообщение от клиента 1: Hi!
Сообщение от клиента 0: What's wrong?
Сообщение от клиента 2: Nothing!
Сообщение от клиента 2: break
Клиент 2 отключился
Клиент 2 подключился
Сообщение от клиента 0: break
Клиент 0 отключился
Сообщение от клиента 2: Today I am here!
Клиент 0 подключился
Сообщение от клиента 0: Beautiful
Сообщение от клиента 2: break
Клиент 2 отключился
Клиент 0 отключился
Сообщение от клиента 1: break
Клиент 1 отключился
```

Запускаем приложения клиентов, при каждом новом подключении сервер выводит сообщение об этом. Далее несколько клиентов отправляют сообщения, могут завершиться, затем вновь возобновить работу.

При этом новому клиенту присваивается первый свободный номер.



```
✝ semestr2/lab6/server  semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Hi!
OK
Введите сообщение:
I am fine. And you?
OK
Введите сообщение:
Hello, client!
OK
Введите сообщение:
What's wrong?
OK
Введите сообщение:
break
OK
✝ semestr2/lab6/server  semestr2_lab6± ./netclient.out localhost 9877
Введите сообщение:
Beautiful
OK
Введите сообщение:
^C
```

Максимально одновременно можно запустить FD_SETSIZE = 256 клиентов. Запустим 5 клиентов одновременно проверим работу.

Сервер:

```
✘ ✝ semestr2/lab6/server  semestr2_lab6± ./netserver.out 9877
Клиент 0 подключился
Клиент 1 подключился
Клиент 2 подключился
Клиент 3 подключился
Клиент 4 подключился
Сообщение от клиента 4: HI!
Сообщение от клиента 0: Hello
Сообщение от клиента 3: Good morning!
Сообщение от клиента 2: Nice to see you!
Сообщение от клиента 1: Excellent!
Сообщение от клиента 2: break
Клиент 2 отключился
Сообщение от клиента 3: break
Клиент 3 отключился
Сообщение от клиента 0: break
Клиент 0 отключился
Сообщение от клиента 1: break
Клиент 1 отключился
Сообщение от клиента 4: break
Клиент 4 отключился
```

Клиенты:

```
semestr2/lab6/server  semestr2_lab6±  ./netclient.out localhost 9877
Введите сообщение:
Good morning!
OK
Введите сообщение:
break
OK
```

```
semestr2/lab6/server  semestr2_lab6±  ./netclient.out localhost 9877
Введите сообщение:
HI!
OK
Введите сообщение:
break
OK
```

```
semestr2/lab6/server  semestr2_lab6±  ./netclient.out localhost 9877
Введите сообщение:
Hello
OK
Введите сообщение:
break
OK
```

```
semestr2/lab6/server  semestr2_lab6±  ./netclient.out localhost 9877
Введите сообщение:
Excellent!
OK
Введите сообщение:
break
OK
```

```
semestr2/lab6/server  semestr2_lab6±  ./netclient.out localhost 9877
Введите сообщение:
Nice to see you!
OK
Введите сообщение:
break
OK
```