Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

Кафедра теоретической прикладной информатики

Лабораторная работа № 2
по дисциплине «Операционные системы, среды и оболочки»

Технология клиент-сервер: эхо-повтор

Факультет:	ПМИ
Группа:	ПМ-92
Бригада:	8
Студенты:	Иванов В., Кутузов И.
Преподаватель:	Кобылянский В. Г.

Новосибирск

2021

Цель работы

Изучить основные принципы разработки клиент-серверных приложений на примере простейшей однопользовательской программы.

Ход работы

Написать простейшее приложение с одним сервером и одним клиентом, используя API-интерфейс низкого уровня.

Вариант 8: Клиент пересылает серверу два числа. Сервер возвращает сумму полученных чисел.

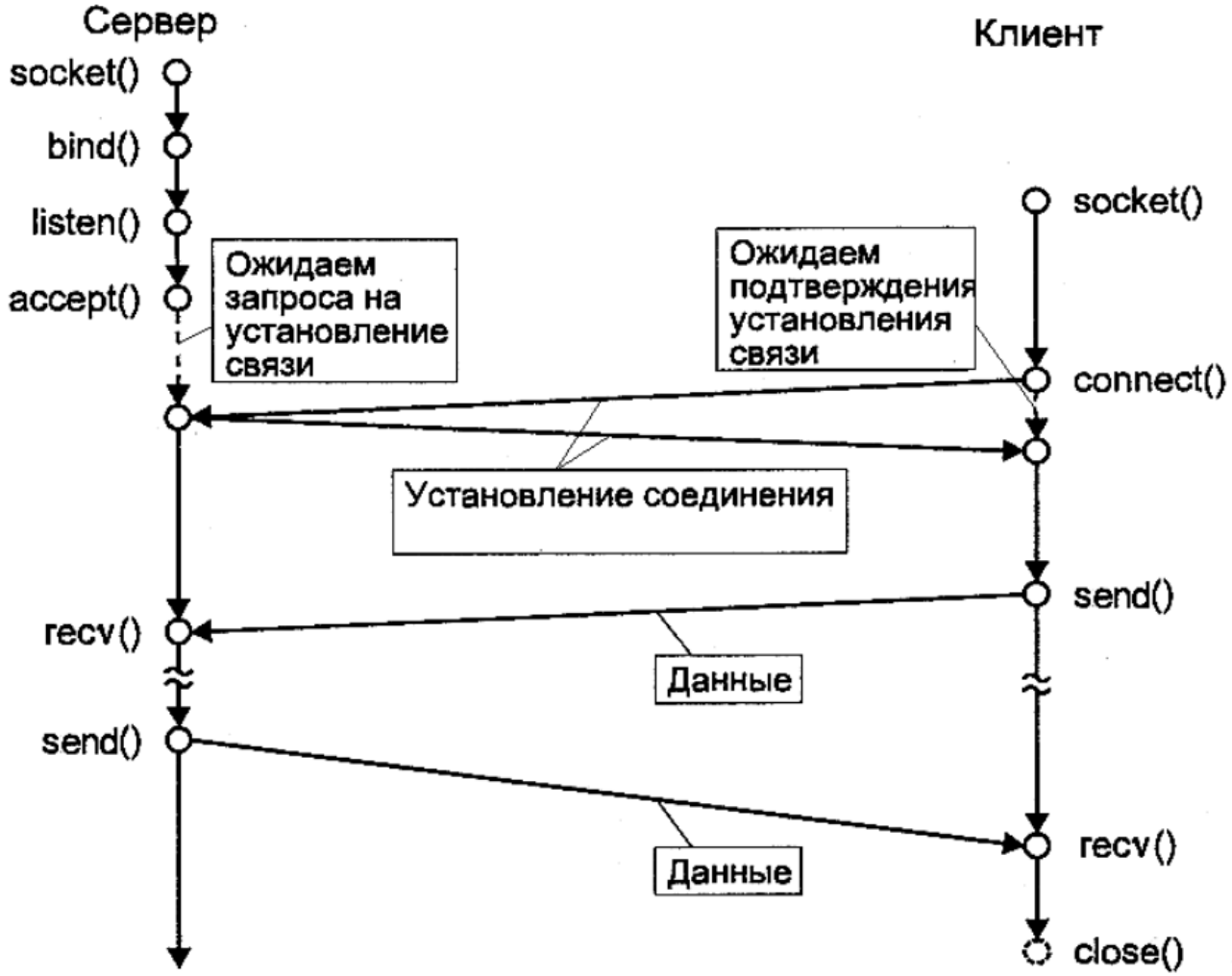

Рис.2.2 Схема установления связи и передачи данных между клиентом и сервером

Первым запускается сервер.

Запуск сервера происходит с указанием номера порта протокола TCP или UDP из диапазона возможных номеров. Каждая бригада создает свой сервер с номером порта, определяемым по правилу: номер бригады + 2000.

В начале программ (как для сервера, так и для клиента) определяются константы - размер буфера и порт:

```
#define BUF_SIZE 100
#define PORT 2008

int main() {
      ...
```

Внутри функции main программы сервера задаются типы переменных и конструкции. Для удобства используется два буфера - для получения от клиента и ответа клиенту:

```
    int s_server, s_new; // дескрипторы сокетов
    int number1, number2, sum; // числа и их сумма
    ...
    char buffer_in[BUF_SIZE] = {0};
    char buffer_out[BUF_SIZE] = {0};
    ...
```

Затем создается сокет на основе протокола TCP и проверяется, не вернула ли функция socket сообщение об ошибке (0):

```
    if ((s_server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) != 0) {
        printf("Socket opened\n");
    }
    else {
        ... // сообщение об ошибке
    }
```

Далее созданный сокет привязывается к порту и проверяется, успешно ли выполнена эта привязка:

```
    if (bind(s_server, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) == SO_ERROR) {
        ... // сообщение об ошибке
    }
```

Сервер начинает свою работу с ожидания запроса от клиента на соединение. Для этого сервер переводится в режим прослушивания порта:

```
    if (listen(s_server, 5) < 0) { // макс. размер очереди запросов на соединение - 5
        ... // сообщение об ошибке
    }
    else {
        printf("Started listening the port %d\n", PORT);
    }
```

Запуск клиента происходит с указанием ему, помимо порта, IP-адреса сервера. Настраиваются порт и IP-адрес сервера, аналогично создается сокет с дескриптором s_client и буфер:

```
    #define SERVER_IP "192.168.0.38"
       ...

    // настройка порта и системы адресации соединений
    serv_addr.sin_port = htons(PORT);
    serv_addr.sin_family = AF_INET;
```

```
// настройка IP-адреса
if(inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr) <= 0) {
    ... // сообщение об ошибке
}
```

Клиент устанавливает связь с сервером и посылает два введенных пользователем числа на сервер:

```
if (connect(s_client, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
    ... // сообщение об ошибке
}

  ...

printf("Enter the number 1: ");
fgets(buffer, BUF_SIZE, stdin); // ввод числа в буфер
send(s_client, buffer, BUF_SIZE, 0); // отправка числа на сервер из буфера

printf("Enter the number 2: ");
fgets(buffer, BUF_SIZE, stdin);
send(s_client, buffer, BUF_SIZE, 0);
```

Сервер, в свою очередь, принимает запрос на установление связи, получает два числа, складывает их и возвращает клиенту результат, снова переходя в состояние ожидания запроса на соединение:

```
s_new = accept(s_server, (struct sockaddr *) &from, (socklen_t *) & from_len);
if (s_new < 0) {
    ... // сообщение об ошибке
}
else {
    ... // сообщение об успешном подключении
}


recv(s_new, buffer_in, BUF_SIZE); // принятие числа в буфер
number1 = atoi(buffer_in); // преобразования строки в целочисленный тип
  ...

recv(s_new, buffer_in, BUF_SIZE);
number2 = atoi(buffer_in);
  ...

sum = number1 + number2;

snprintf(buffer_out, BUF_SIZE, "%d", sum); // преобразование числа в строку
send(s_new, buffer_out, BUF_SIZE, 0); // отправка клиенту
```

Наконец, клиент принимает результат и на этом работа завершается:

```
recv(s_client, buffer, BUF_SIZE);
```

Код программ

server.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <arpa/inet.h>
#define BUF_SIZE 100
#define PORT 2008

int main() {

    int s_server, s_new; // sockets
    int number1, number2, sum; // numbers to add and their sum
    struct sockaddr_in serv_addr;
    struct sockaddr_in from;
    char buffer_in[BUF_SIZE] = {0};
    char buffer_out[BUF_SIZE] = {0};
    int from_len = sizeof(from);

    // creating socket
    if ((s_server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) != 0) {
        printf("Socket opened\n");
    }
    else {
        printf("Error: Socket was not opened!\n");
        exit(EXIT_FAILURE);
    }

    // socket parameters
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    // bind server socket to specific port
    if (bind(s_server, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) == SO_ERROR) {
        printf("Error: Socket was not bound!\n");
        printf("Socket closed\n");
        exit(EXIT_FAILURE);
    }

    // starting socket listening
    if (listen(s_server, 5) < 0) { // max queue size = 5
        printf("Error!\n");
        printf("Socket closed\n");
        exit(EXIT_FAILURE);
    }
    else {
        printf("Started listening the port %d\n", PORT);
    }

    // accepting a client connection
    s_new = accept(s_server, (struct sockaddr *) &from, (socklen_t *) & from_len);
    if (s_new < 0) {
        printf("Error: Connection was not accepted!\n");
        printf("Socket closed\n");
        exit(EXIT_FAILURE);
    }
```

```
    else {
        printf("Connection accepted from %s (port: %d)\n", inet_ntoa(from.sin_addr),
htons(from.sin_port));
    }

    // receiving messages from the client
    recv(s_new, buffer_in, BUF_SIZE);
    number1 = atoi(buffer_in);
    printf("Message received: %d\n", number1);

    recv(s_new, buffer_in, BUF_SIZE);
    number2 = atoi(buffer_in);
    printf("Message received: %d\n", number2);

    sum = number1 + number2;

    // sending to the client
    snprintf(buffer_out, BUF_SIZE, "%d", sum);
    send(s_new, buffer_out, BUF_SIZE, 0);
    printf("Answer sent\n");

    // closing server socket
    close(s_new);
    printf("Connection closed\n");

    close(s_server);
    printf("Socket closed\n");

    return 0;
}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <arpa/inet.h>
#define BUF_SIZE 100
#define PORT 2008
#define SERVER_IP "192.168.0.38"

int main() {

    int s_client; // socket
    struct sockaddr_in serv_addr;
    char buffer[BUF_SIZE] = {0};

    // creating socket
    if ((s_client = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Error: Socket was not opened!\n");
        printf("Socket closed\n");
        exit(EXIT_FAILURE);
    }

    printf("Socket opened\n");
```

```
    // set server port
    serv_addr.sin_port = htons(PORT);
    serv_addr.sin_family = AF_INET;

    // set IP
    if(inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr) <= 0) {
        printf("Error: Invalid IP-adress!");
        exit(EXIT_FAILURE);
    }

    // connecting to the server
    if (connect(s_client, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        printf("Error: Connection was not established!\n");
        printf("Socket closed\n");
        exit(EXIT_SUCCESS);
    }

    printf("Connected to the server\n");

    // sending messages to the server
    printf("Enter the number 1: ");
    fgets(buffer, BUF_SIZE, stdin);
    send(s_client, buffer, BUF_SIZE, 0);
    printf("Message sent\n");

    printf("Enter the number 2: ");
    fgets(buffer, BUF_SIZE, stdin);
    send(s_client, buffer, BUF_SIZE, 0);
    printf("Message sent\n");


    // receiving messages from the server
    recv(s_client, buffer, BUF_SIZE);
    printf("Answer received: %s\n", buffer);

    // closing socket
    close(s_client);

    printf("Socket closed\n");

    return 0;
}
```

Тестирование

Программы сервера и клиента были скомпилированы при помощи компилятора gcc и запущены на разных компьютерах с операционной системой Linux

Запуск сервера:

```
Socket opened
Started listening the port 2008
```

Запуск клиента:

```
Socket opened
Connected to the server
Enter the number 1:
```

Реакция сервера:

```
Socket opened
Started listening the port 2008
Connection accepted from 192.168.0.40 (port: 40796)
```

Клиент последовательно вводит два числа:

```
Socket opened
Connected to the server
Enter the number 1: 17
Message sent
Enter the number 2: 3
Message sent
Answer received: 20
Socket closed
```

Сервер принимает числа и возвращает результат, а после прерывания соединения закрывает сокеты:

```
Socket opened
Started listening the port 2008
Connection accepted from 192.168.0.40 (port: 40796)
Message received: 17
Message received: 3
Answer sent
Connection closed
Socket closed
```

IP-адрес сервера: **192.168.0.38**

```
~/university-tasks/Computer Networks/Client-Server Model  master  ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 66:5d:86:5c:a4:8b brd ff:ff:ff:ff:ff:ff permaddr 80:30:49:e9:76:6d
    inet 192.168.1.102/24 brd 192.168.1.255 scope global wlp1s0
       valid_lft forever preferred_lft forever
    inet 192.168.0.38/24 brd 192.168.0.255 scope global dynamic noprefixroute wlp1s0
       valid_lft 13580sec preferred_lft 11780sec
    inet6 fe80::52c7:8d63:36c1:46e0/64 scope link
       valid_lft forever preferred_lft forever
```

Вывод

В ходе выполнения лабораторной работы были изучены базовые функции для работы с сокетами, на основе которых было построено простейшее приложение модели ‘Клиент-сервер’.