

Лабораторная работа №10 – Создание соединений на сокетах

Цель работы

Освоение набора системных вызовов для создания сокетных соединений различных типов для обмена данными по сети.

Пункт 1

Скомпилируйте и выполните программу `socketpair.cpp`, иллюстрирующую создание простейшего вида сокета и обмен данными двух родственных процессов. Проанализируйте вывод на консоль. Существует ли зависимость обмена от различных соотношений величин временных задержек (в вызовах `sleep()`) в процессе-родителе и в процессе-потомке?



```
wooffle@PC:~/spbstu-os-labs/labs/lab10/src1$ make
g++ socketpair.cpp -o socketpair
wooffle@PC:~/spbstu-os-labs/labs/lab10/src1$ ./socketpair
p->c:0
c->p: 1
p->c:2
c->p: 3
p->c:4
c->p: 5
p->c:6
c->p: 7
p->c:8
c->p: 9
wooffle@PC:~/spbstu-os-labs/labs/lab10/src1$
```

Рис. 10-1 Компиляция и исполнение программы `socketpair`.

В результате на консоль получили цифры от 0 до 9. В ходе выполнения программы была создана пара сокетов. После создания потомка стоит ветвление для обработки родителя и потомка отдельно. Закрываются ненужные сокеты. Родитель начинает отправлять числа потомку, потомок их читает, выводит на консоль и отправляет своё число родителю и так до 9.

Изменение величины в `sleep()` не влияет на функционал программы. Будет изменяться задержка перед следующим числом, т.к. процессы ждут друг от друга сообщений по сокетам и не идут дальше.

Пункт 2

Скомпилируйте программы `echo_server.cpp` и `echo_client.cpp`, задавая им при компиляции разные имена (размещаем файлы в одном каталоге). Запустите программы сервера и клиента на разных терминалах. Введите символьную информацию в окне клиента и проанализируйте вывод. Какой разновидности

принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия? С чем связано создание специального файла в текущем каталоге во время исполнения программ?

```
wooffie@PC:~/spbstu-os-labs/labs/lab10/src2$ lsof echo_socket
COMMAND  PID    USER  FD  TYPE             DEVICE SIZE/OFF  NODE NAME
server   6109  wooffie  3u  unix 0x0000000000000000      0t0 57313 echo_socket type=STREAM
server   6109  wooffie  4u  unix 0x0000000000000000      0t0 57314 echo_socket type=STREAM
```

Рис. 10-2 Специальный файл-сокет.

При выполнении программы создаём сокет-файл (локальный сокет). Сокеты в файловом пространстве имён используем в качестве адресов имена файлов специального типа.

```
wooffie@PC:~/spbstu-os-labs/labs/lab10/src2$ ./server
Waiting for a connection...
Connected.
Waiting for a connection...
Connected.

wooffie@PC:~/spbstu-os-labs/labs/lab10/src2$ ./client
Trying to connect...
Connected.
> Hi
echo> Hi
> os bubuntu
echo> os bubuntu
> lenght test-----
echo> lenght test-----
> ^X^C
wooffie@PC:~/spbstu-os-labs/labs/lab10/src2$ ./client
Trying to connect...
Connected.
> hi again
echo> hi again
>
[0] 0:./client* "PC" 13:09 08-ОКТ-21
```

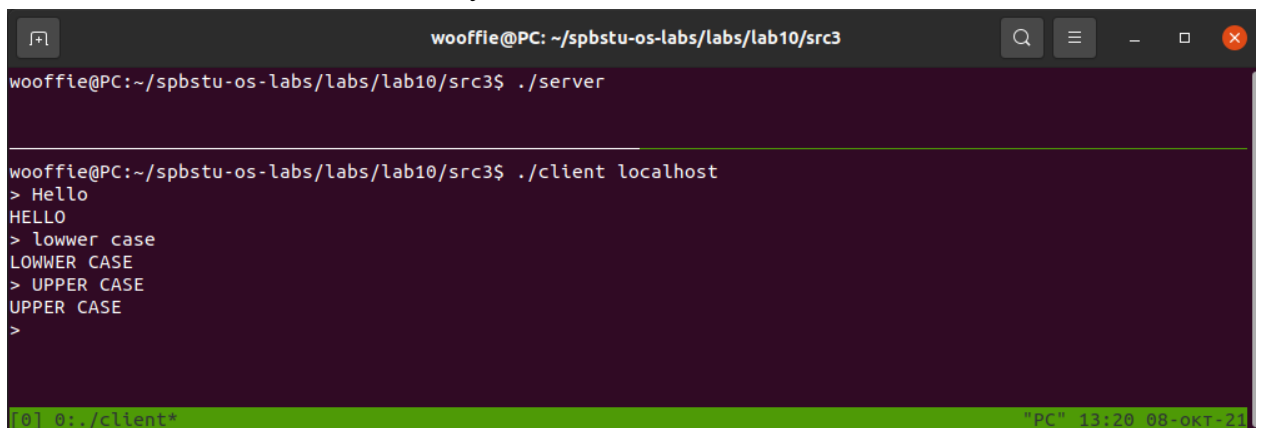
Рис. 10-3 Работа программ второго пункта.

Клиент получает от сервера вывод, который копирует его ввод. Данный пример иллюстрирует **сокеты поточного вида**. Если отключиться со второго терминала, то сервер будет ждать следующего клиента.

Пункт 3

Скомпилируйте с разными именами программы `sock_c_i_srv.cpp` и `sock_c_i_clt.cpp` (в них используется общий include файл `local_c_i.h`). Запустите программы сервера и клиента на разных терминалах. При запуске клиента указывайте в качестве параметра командной строки имя хоста `localhost`. Введите символьную информацию в окне клиента и поясните вывод. Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия?

Выводом является строка клиента в верхнем регистре. Со стороны сервера вывод на консоль не используется.



```
wooffie@PC: ~/spbstu-os-labs/labs/lab10/src3
wooffie@PC:~/spbstu-os-labs/labs/lab10/src3$ ./server
wooffie@PC:~/spbstu-os-labs/labs/lab10/src3$ ./client localhost
> Hello
HELLO
> lower case
LOWER CASE
> UPPER CASE
UPPER CASE
>
```

Рис. 10-4 Сервер-клиент из третьего пункта.

Сокеты в примере относятся к **сетевым**, так как требуется присваивание сетевого адреса (в нашем случае localhost) и никаких файлов не создаётся (в случае UNIX-сокеты.)

Пункт 4

Модифицируйте программу echo_server.cpp так, чтобы при ответе на запросы клиента что-либо выводилось в окне сервера.

Для вывода в окне сервера были добавлены функции файла "echo_server.cpp". Листинг представлен ниже:

Листинг 10-1 Изменения в исходном коде сервера
<pre>/* ** echo_server.cpp -- the echo server for echo_cient.cpp; demonstrates UNIX sockets */ #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <errno.h> #include <string.h> #include <sys/types.h> #include <sys/socket.h> #include <sys/un.h> #define SOCK_PATH "echo_socket" int main(void) { int s, s2, t, len; struct sockaddr_un local, remote; char str[100]; if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) { perror("socket"); exit(1);</pre>

```

    }
    local.sun_family = AF_UNIX;
// remote.sun_family = AF_UNIX;    //
    strcpy(local.sun_path, SOCK_PATH);
    unlink(local.sun_path);
    len = strlen(local.sun_path) + sizeof(local.sun_family);
    if (bind(s, (struct sockaddr *)&local, len) == -1) {
        perror("bind");
        exit(1);
    }
    if (listen(s, 5) == -1) {
        perror("listen");
        exit(1);
    }
    for(;;) {
        int done, n;
        printf("Waiting for a connection...\n");
        t = sizeof(remote);
        if ((s2 = accept(s, (struct sockaddr *)&remote, (socklen_t *)&t)) == -1) {
            perror("accept");
            exit(1);
        }
        printf("Connected.\n");
        done = 0;
        do {
            n = recv(s2, str, 100, 0);
            printf("client -> %s",str);
            if (n <= 0) {
                if (n < 0) perror("recv");
                done = 1;
            }
            if (!done)
                printf("client <- %s",str);
            if (send(s2, str, n, 0) < 0) {
                perror("send");
                done = 1;
            }
        } while (!done);
        close(s2);
    }
    return 0;
}

```

Теперь сервер будет выводить - какую информацию он получил, а какую вернул клиенту:

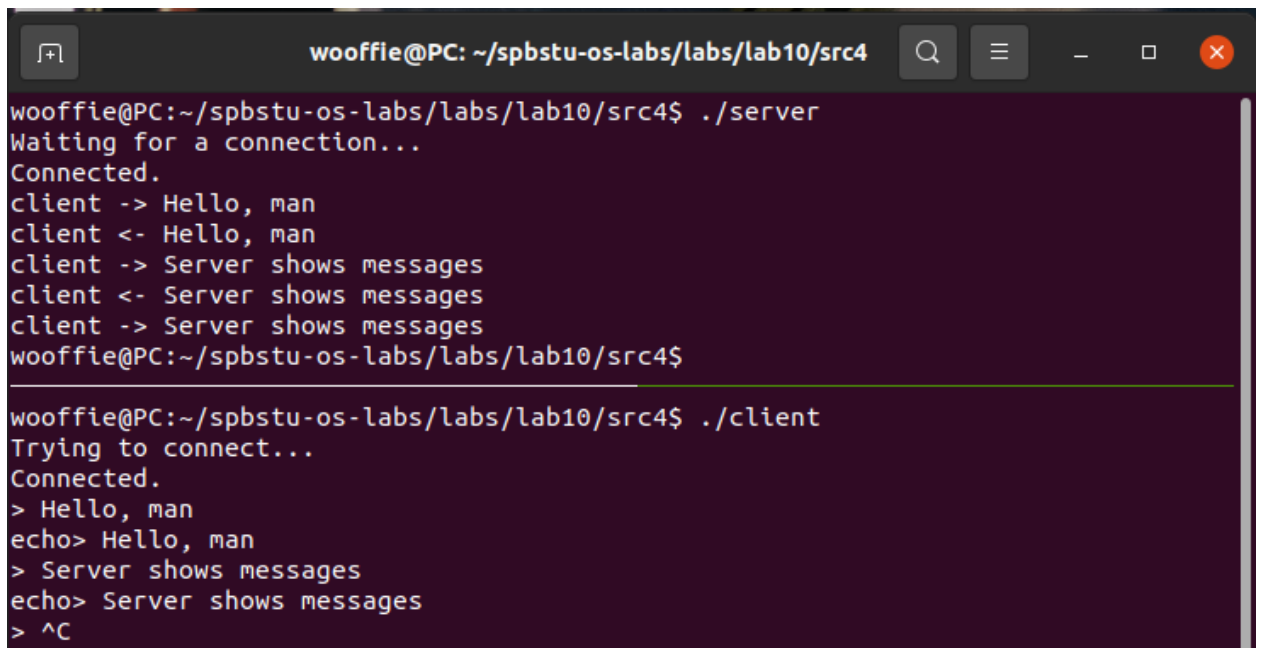
A terminal window titled 'wooffie@PC: ~/spbstu-os-labs/labs/lab10/src4'. The prompt is 'wooffie@PC:~/spbstu-os-labs/labs/lab10/src4\$'. The user enters './server', and the output is 'Waiting for a connection...', 'Connected.', 'client -> Hello, man', 'client <- Hello, man', 'client -> Server shows messages', 'client <- Server shows messages', 'client -> Server shows messages', and the prompt returns. Then the user enters './client', and the output is 'Trying to connect...', 'Connected.', '> Hello, man', 'echo> Hello, man', '> Server shows messages', 'echo> Server shows messages', and finally '> ^C'.

Рис. 10-5 Изменённая программа сервера.

Пункт 5

Испытайте работу эхо-сервера при одновременной работе с несколькими клиентами.

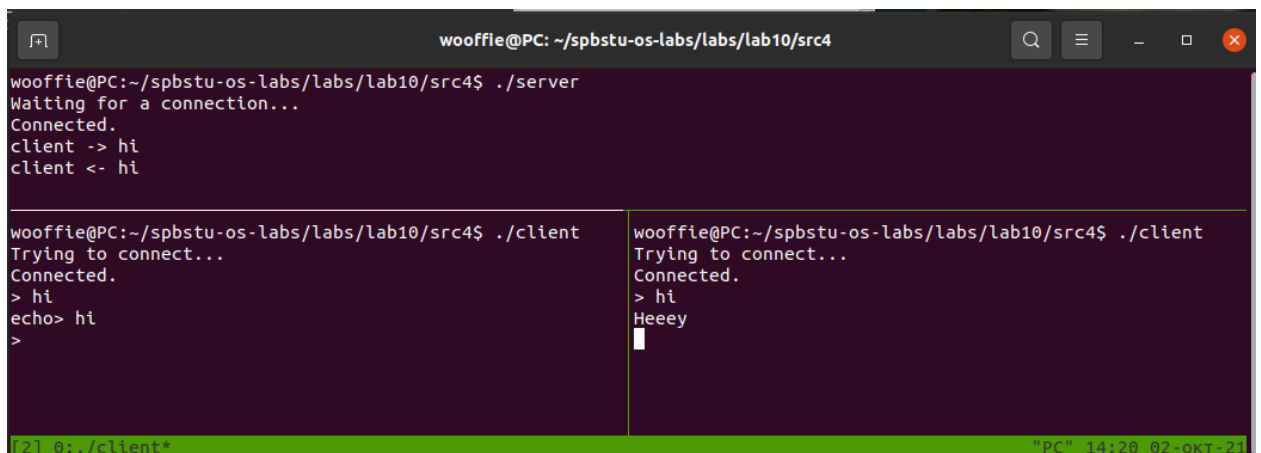
A terminal window titled 'wooffie@PC: ~/spbstu-os-labs/labs/lab10/src4'. The prompt is 'wooffie@PC:~/spbstu-os-labs/labs/lab10/src4\$'. The user enters './server', and the output is 'Waiting for a connection...', 'Connected.', 'client -> hi', and 'client <- hi'. Then the user enters './client', and the output is 'Trying to connect...', 'Connected.', '> hi', and 'echo> hi'. A second terminal window is shown to the right, also titled 'wooffie@PC: ~/spbstu-os-labs/labs/lab10/src4', with the prompt 'wooffie@PC:~/spbstu-os-labs/labs/lab10/src4\$'. The user enters './client', and the output is 'Trying to connect...', 'Connected.', '> hi', and 'Heeey'. The bottom status bar shows '[2] 0:./client*' and '"PC" 14:20 02-окт-21'.

Рис. 10-6 Подключение нескольких клиентов.

Можно сделать вывод, что данный эхо-сервер не предназначен для подключения нескольких клиентов. Чтобы он мог работать с несколькими клиентами необходимо добавить или fork или exes.

Возьмём метод создание потомка fork(). Для каждого клиента будем делать свой дочерний процесс сервера. Вот что получилось

```
wooffie@PC: ~/spbstu-os-labs/labs/lab10/src5
wooffie@PC:~/spbstu-os-labs/labs/lab10/src5$ ./server
Client 4 connected
client 4 -> Hi server
client 4 <- Hi server
Client 5 connected
client 5 -> Hello, guys!
client 5 <- Hello, guys!
Client 6 connected
client 6 -> Hi, to all
client 6 <- Hi, to all

wooffie@PC:~/spbstu-os-labs/labs/lab10/src5$ ^C
wooffie@PC:~/spbstu-os-labs/labs/lab10/src5$ ./client
Trying to connect...
Connected.
> Hi server
echo> Hi server
>

wooffie@PC:~/spbstu-os-labs/labs/lab10/src5$ ./client
Trying to connect...
Connected.
> Hello, guys!
echo> Hello, guys!
>

wooffie@PC:~/spbstu-os-labs/labs/lab10/src5$ ./client
Trying to connect...
Connected.
> Hi, to all
echo> Hi, to all
>

[5] 0:./client*
```

Рис. 10-7 Модифицированная программа сервера.

Вывод

Был освоен набор системных вызовов для создания сокетных соединений. Были изучены различные сокетные соединения и их особенности. Также изучены особенности сервера, который обрабатывает запросы клиентов, и в его исходный код привнесены изменения для более комплексной демонстрации его работы.