**Лабараторная 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?



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 велечины в sleep() не влияет на функционал программы. Будет изменятся 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?

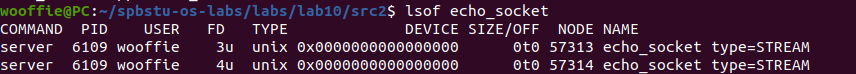


Рис. 10-2 Специальный файл-сокет.

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

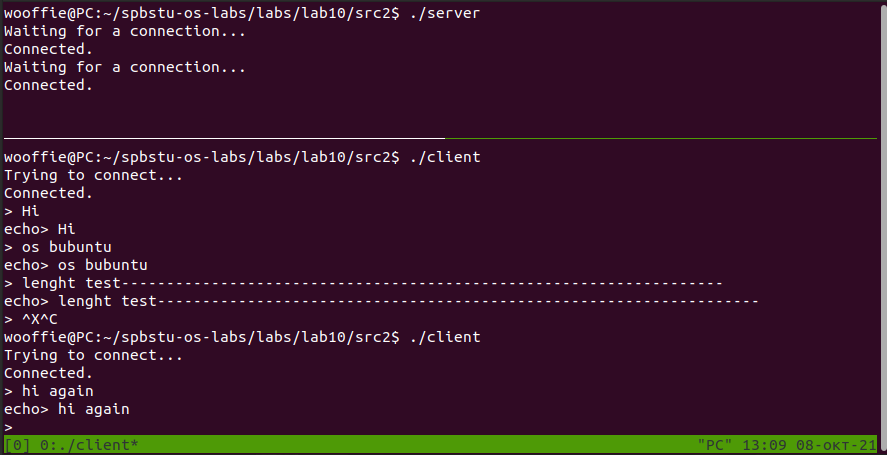


Рис. 10-3 Работа программ второго пункта.

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.

**Пукнт 3**

Скомпилируйте c 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.

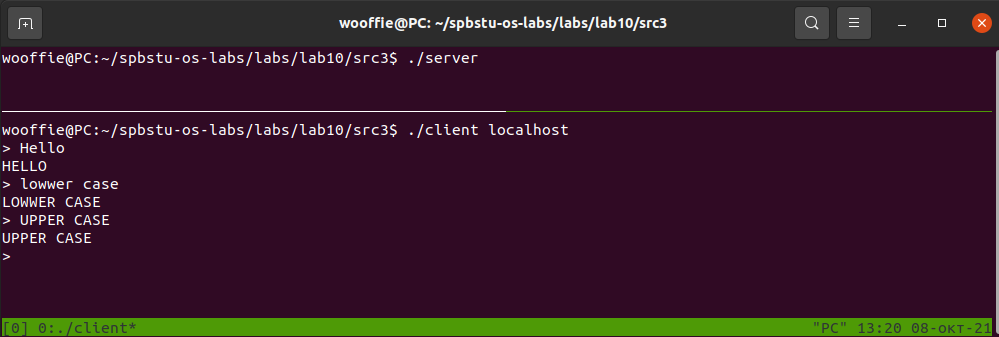


Рис. 10-4 Сервер-клиент из третьего пункта.

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

**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 |
| /\*  \*\* 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);  }  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:

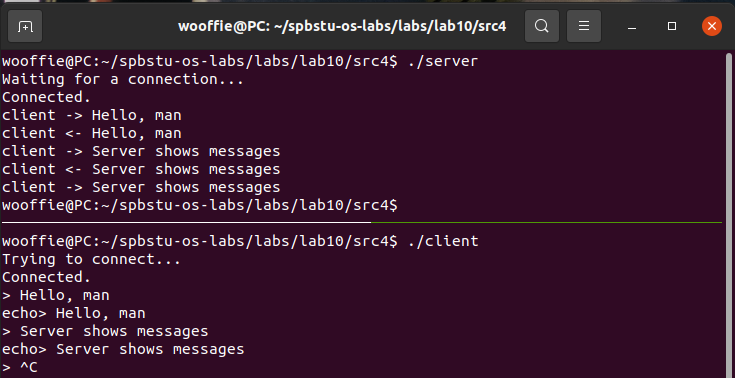


Рис. 10-5 Изменённая программа сервера.

**Пункт 5**

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

Изображение выглядит как текст, снимок экрана, экран

Автоматически созданное описание

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 exec.

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

Изображение выглядит как текст

Автоматически созданное описание

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.