Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
Институт заочно-вечернего обучения

Допускаю к защите
Руководитель _____ Ю. Р. Басиров
подпись, И. О. Фамилия

Доступ к приложению через socket
Наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к лабораторной работе по дисциплине

«Технология программирования»

1.011.00.00 ПЗ
обозначение документа

Разработал студент группы ЭВМбз-16-1 _____ А. А. Михиденко
подпись И. О. Фамилия

Нормоконтроль _____ Ю. Р. Басиров
подпись И. О. Фамилия

Лабораторная работа защищена с оценкой _____

Иркутск 2021

Содержание

1 Теоретический материал

Socket - это программный интерфейс, который функционирует на верхнем уровне стека сетевого протокола TCP/IP.

Впервые интерфейс был описан в BSD Unix. Применяется для обмена данными между сетевыми приложениями, либо между модулями одного такого приложения. Структура и методы передачи определяются индивидуально исходя из поставленных задач и целей, и описываются стандартом POSIX.1

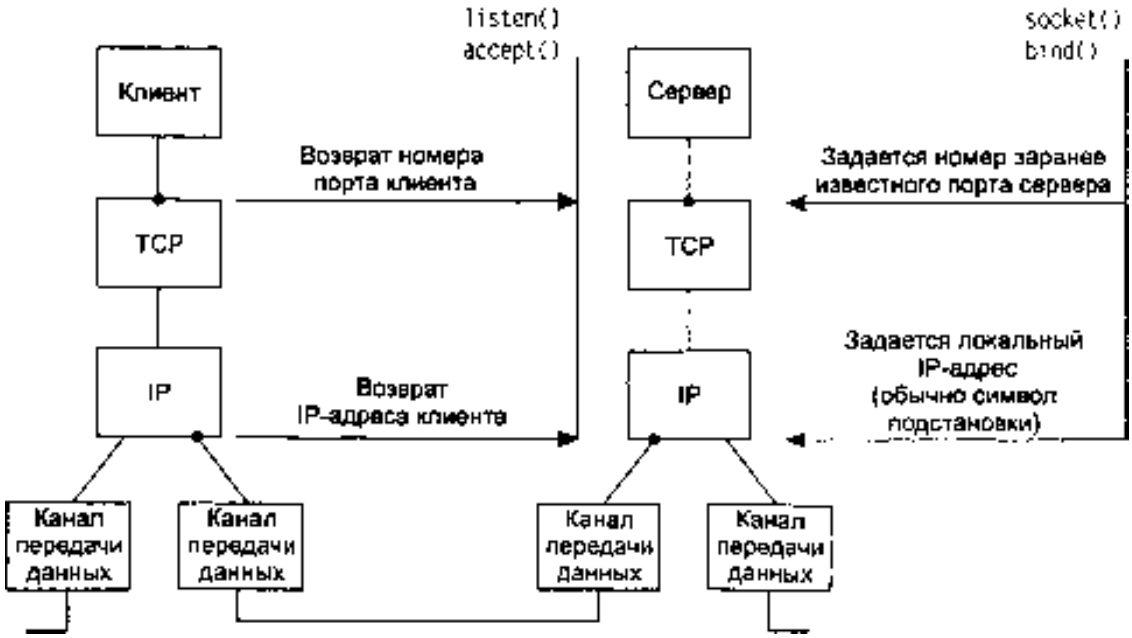

Рисунок 1 - Схема организации канала связи через socket

Прежде чем клиент и сервер TCP смогут взаимодействовать друг с другом, каждый из них должен определить пару сокетов для соединения: локальный IP-адрес, локальный порт, удаленный IP-адрес, удаленный порт. На рис. 1 мы схематически изображаем эту принципиальную модель.

Режимы работы сокета разделяются на серверные и клиентские, либо комбинированные. В случае сервера: сокет открывается, затем происходит его назначение, после чего он переход в состояние прослушивания. В случае обнаружения входящего запроса, сокет может согласится на соединение с клиентом, он отправляет или получает данные, после чего сокет закрывается.

2 Постановка задачи

Задача: организовать прямой доступ к консоли игрового сервера при помощи программного интерфейса(socket).

Цель: применить на практике навыки работы с программным интерфейсом(socket).

В качестве консоли игрового сервера выступает приложение ds от компании valve. При помощи доступа к API организованного модификацией, которая позволяет перехватывать виртуальные функции движка, я организую ПИ который будет прослушивать входящие соединения на заданном интерфейсе. И согласно логике ответа, будет отправлять данные запрашиваемому клиенту.

3 Выполнение задания

Ссылка на проект:

И так, для того что бы определит сокет в нашей среде, необходимо создать переменную которая будет хранить указатель на сокет(сервера):

Socket sock;

после чего надо вызвать метод создания экземпляра класса и определить для него указатель:

sock = new Socket(SOCKET_TCP, OnSocketError);

За регистрацию входящего соединения отвечает метод обратного вызова Listen, для которого определен метод OnSocketIncoming, который принимает следующие фактические параметры:

- *socket*, указатель на прослушиваемый сокет
- *newSocket*, потомок прослушиваемого сокета
- *remoteIP*
- *remotePort*

В данном случае потомок сокета используется для отправки данных клиенту, по сколько указатель на прослушиваемый сокет будет удален.

Метод обратного вызова OnSocketIncoming, в свою очередь, вызывает еще несколько под-методов обратного вызова:

- *SetReceiveCallback*, запрос успешен
- *SetDisconnectCallback*, клиент отключился
- *SetErrorCallback*, ошибка запроса

В нашем случае всю основную операцию обработки вызова определяет метод SetReceiveCallback. Давайте рассмотрим его подробнее.

У данного запроса так же есть свой определенный тип данных, и важным здесь является фактический параметр receiveData. Зная базовые операции работы со строками в C, можем определить логику метода.

К примеру, команда exit должна сразу быть заблокирована. По причине того что доступ к консоли сервера могут иметь неадекватные люди - страхуем сервер от неожиданного завершения работы:

```
if(StrContains(receiveData, "exit", false) == 0) {
	CloseHandle(socket);
	return;
}
```

По сути это уже объясняет весь принцип работы ПИ для консоли игрового сервера. Но если вам этого мало, давайте продолжим.

Определим переменную для хранения символа ch, в нее мы будем помещать результат метода StrContains, вызванного для поиска подстроки excmd. Данная подстрока является указателем на то, что после нее следует команда сервера, которая будет отправлена в его консоль.

```
if((ch = StrContains(receiveData, "excmd", false)) != -1) {
  char buffer[4096];
  ServerCommandEx(buffer, sizeof(buffer), receiveData[ch+6]);
}
```

Переменная buffer создана для хранения результата ответа игрового сервера, которую мы отправляем обратно клиенту.

Финальным штрихом будет отправка этого ответа, и удаление указателя на вызывающий сокет.

```
socket.Send(buffer, strlen(buffer));
```

Описание front-end

Реализация пользовательского интерфейса выглядит следующим образом: из пунктов управления сервером выбирается один "Консоль сервера", которая предоставляет нам простую форму ввода и вывода информации, а так же кнопку отправки данных на удаленный сервер, рис. 2.

Рисунок 2 - Внешний вид панели управления сервером

При нажатии кнопки Выполнить происходит отправка данных на указанный. Данные передаются на строну BackEnd, где предварительно обрабатываются и по результатам успешной проверки, отправляются на удаленный игровой сервер.

Для отправки с front используется JS (AJAX), для back - php.

Фрагменты кода back

После предварительной обработки данных $_POST запроса вызывается метод класса http_Exec:

```
$data = $ServerControls->http_Exec(
 '192.168.88.241:'.$res[0]['port'], 'post', ['excmd' => $data['cmd']]
);
```

Определяем необходимые заголовки для отправки правильного POST запроса к сокету удаленного игрового сервера:

```
$out  = "POST {$path} HTTP/1.1\r\n";
$out .= "Accept: */*\r\n";
$out .= "Accept-Language: ru-ru\r\n";
$out .= "Content-Type: text/html; charset=UTF-8\r\n";
$out .= "User-Agent: {$_SERVER['HTTP_USER_AGENT']}\r\n";
$out .= "Host: {$host}\r\n";
$out .= 'Content-Length: '.strlen($parameter)."\r\n";
$out .= "Connection: Close\r\n\r\n";
$out .= $parameter;
```

При помощи метода fsockopen отдаем данные серверу, читаем результат и передаем ответ обратно обработчику запроса на AJAX:

```
$fp = fsockopen($host, $port, $errno, $errstr, 3);
!$fp && die();
$bt = fwrite($fp, $out);
if($bt)
{
  $data = PHP_EOL; $len = strlen($data);
  while(!feof($fp))
  {
    $data .= fgets($fp, 4096);
  }
}
fclose($fp);
return strlen($data) > $len ? $data : '';
```

Фрагмент кода JS, который восполняет отправку запроса и обработку результата возвращаемых данных:

```
$.post('./app/modules/module_page_server_controls/includes/ServerJS.php', {
  'server-exec': { sid: sid, cmd: data }, 'auth' : '<?=$_SESSION['steamid32']?>'
}).done(function(ajaxResponse) {
  $("#exec-wait").empty().append('команда успешно выполнена!');
  $("#exec-wait").css('color', '#00ff37f5');
  $("#console").css('background-color', bgcol);
  $("#console").text(ajaxResponse).scrollBottom();
}).fail(function() {
  $("#exec-wait").empty().append('ОШИБКА ВЫПОЛНЕНИЯ КОМАНДЫ!');
  $("#console").css('background-color', bgcol);
  $("#exec-wait").css('color', '#ffe000f5');
}).always(function() {
  setTimeout(function () {$("#exec-wait").empty();}, 1500);
  ajaxExecRequest = false;
});
```

Вывод

В данной ЛР отражена простая методика использования программных интерфейсов, с принципом работы которой я ознакомился.

Можно сказать что технология программных интерфейсов расширяет возможности взаимодействия между пользователем и сервисом. Технология проста реализации и имеет богатую поддержку со стороны разработчиков.