МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматики и вычислительной техники

Кафедра АСУ



**ОТЧЕТ**

по лабораторной работе № 4«СЕТЕВОЕ ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ СОКЕТОВ»

Дисциплина «Технологии и методы программирования», 4 семестр

Выполнили:

студент группы АВТ-214

Крипаков Леонид Дмитриевич  
Саушкин Никита Максимович

г. Новосибирск

2024 год

**Постановка задачи.**

Задание 1: Разработать программу-клиент, которая должна:

− запрашивать у пользователя адрес программы-сервера;

− устанавливать соединение с сервером;

− передавать на сервер имя выбранного каталога;

− принимать ответ от сервера и выводить его на экран;

− закрывать соединение с сервером.

Задание 2: Разработать программу-сервер, которая должна:

− ожидать запрос от программ-клиентов на соединение;

− устанавливать соединение с клиентами, при установлении соединения передавать список логических устройств клиенту;

− принимать от клиента данные;

− если это имя каталога, передавать клиенту структуру каталога (имена файлов и подкаталогов);

− если это имя текстового файла, передавать клиенту содержимое файла.

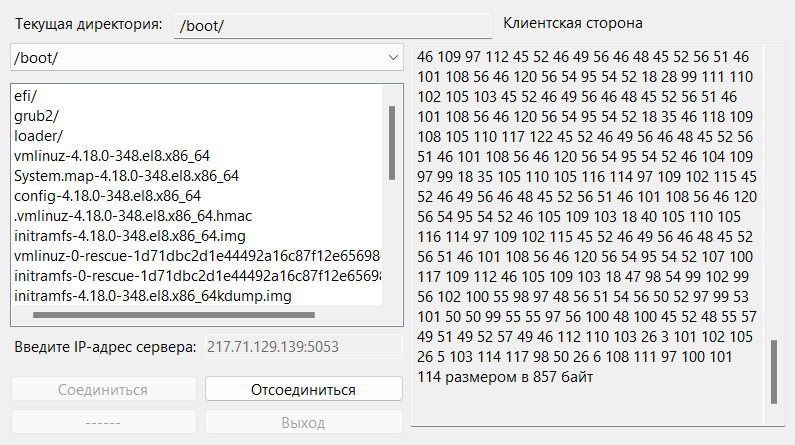
− закрывать соединение с клиентом при получении уведомления.

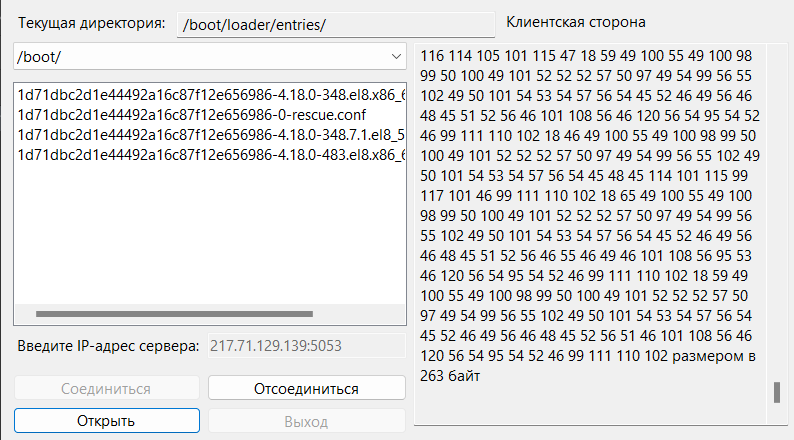
**Исходный код класса и тестирующей программы, оформленный в соответствии с приложением А.**

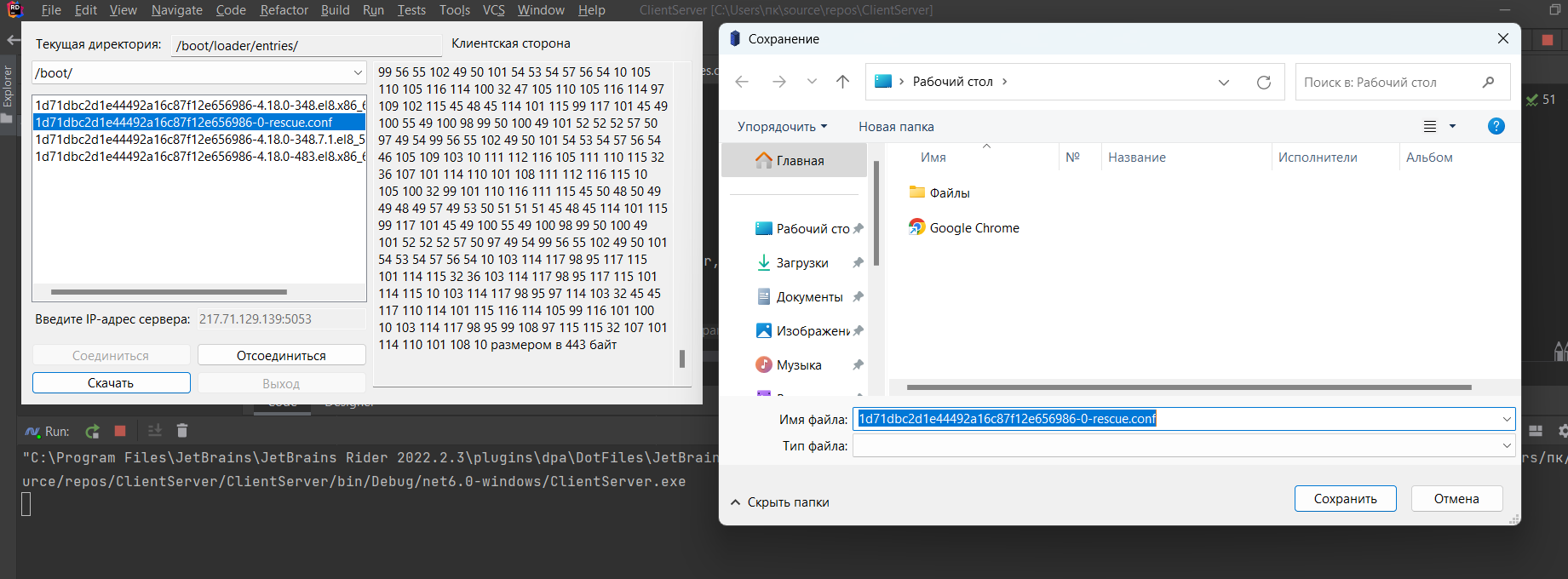
|  |
| --- |
| **Сервер** |
| use std::{  io::{prelude::\*, BufReader, ErrorKind},  time::Duration,  };  use log::{debug, error, info};  use prost::Message;  use tokio::net::{TcpListener, TcpStream};  pub mod tcp\_entities {  include!("../tcp\_entities.rs");  }  use tcp\_entities::\*;  use tokio::runtime;  use tokio::{runtime::Runtime, task};  const SERVER\_ADDR: &str = "172.17.10.110:7878";  fn main() {  env\_logger::init();  //prost\_build::Config::new()  //.protoc\_arg("--experimental\_allow\_proto3\_optional")  //.compile\_protos(&["./entities.proto"], &["./"])  //.unwrap();  let rt = runtime::Builder::new\_current\_thread()  .enable\_io()  .build()  .unwrap();  rt.block\_on(async {  let listener = TcpListener::bind(SERVER\_ADDR).await.unwrap();  info!("Server started on {}", SERVER\_ADDR);  loop {  let (stream, \_) = listener.accept().await.unwrap();  info!("New connection with {}", &stream.peer\_addr().unwrap());  task::spawn(async move {  handle\_connection(stream).await.unwrap();  });  }  });  }  async fn handle\_connection(stream: TcpStream) -> Result<(), std::io::Error> {  let mut buffer = [0; 1024];  loop {  stream.readable().await?;  match stream.try\_read(&mut buffer) {  Ok(0) => {  info!("Connection with {} terminated", stream.peer\_addr().unwrap());  return Ok(());  }  Ok(bytes\_read) => {  let mut req = Request::default();  if let Err(e) = req.merge(&buffer[..bytes\_read]) {  info!("Err: {}", e);  continue;  }  info!("{:?}", req);  let resp = match req.action\_type() {  request::ActionType::List => list\_dir(req),  request::ActionType::DownloadChunk => download\_chunk(req),  request::ActionType::FileInfo => read\_file(req),  };  let mut resp\_bytes = Vec::with\_capacity(resp.encoded\_len());  resp.encode(&mut resp\_bytes).unwrap();  stream.writable().await?;  stream.try\_write(&resp\_bytes[..])?;  }  Err(e) if e.kind() == std::io::ErrorKind::WouldBlock => {  continue;  }  Err(e) => {  info!("Error reading from socket: {}", e);  return Err(e);  }  }  }  }  fn handle\_error(err: Option<std::io::Error>) -> Result<(), response::Status> {  if err.is\_none() {  return Ok(());  }  let status = match err.unwrap().kind() {  ErrorKind::NotFound => response::Status::NotFound,  ErrorKind::PermissionDenied => response::Status::PermissionDenied,  \_ => response::Status::Error,  };  Err(status)  }  fn list\_dir(req: Request) -> Response {  let mut resp = Response::default();  let mut directory = Directory::default();  directory.path = req.path;  if let Err(status) = handle\_error(std::fs::read\_dir(&directory.path).err()) {  resp.set\_status(status);  return resp;  }  for entry in std::fs::read\_dir(&directory.path).unwrap() {  let entry = entry.as\_ref().unwrap();  let name = entry.file\_name().into\_string().unwrap();  let file\_type = entry.file\_type().unwrap();  if file\_type.is\_dir() {  directory.subdirectories.push(name);  } else if file\_type.is\_file() {  directory.files.push(name);  }  }  resp.set\_status(response::Status::Ok);  resp.resource = Some(response::Resource::Directory(directory));  resp  }  fn download\_chunk(req: Request) -> Response {  let mut resp = Response::default();  let mut chunk = FileChunk::default();  chunk.offset = req.offset.unwrap();  let mut file = std::fs::File::open(&req.path).unwrap();  let seek\_pos = req.offset.unwrap() \* req.chunk\_size.unwrap();  file.seek(std::io::SeekFrom::Start(seek\_pos as u64)).unwrap();  let mut buffer = vec![0u8; req.chunk\_size.unwrap() as usize];  file.read(&mut buffer).unwrap();  chunk.content = buffer.into();  resp.set\_status(response::Status::Ok);  resp.resource = Some(response::Resource::Chunk(chunk));  resp  }  fn read\_file(req: Request) -> Response {  let mut resp = Response::default();  let mut file = File::default();  let mut path = std::path::PathBuf::from(".");  path.push(&req.path);  if let Err(status) = handle\_error(std::fs::read(&req.path).err()) {  resp.set\_status(status);  return resp;  }  file.name = path.file\_name().unwrap().to\_str().unwrap().to\_string();  file.size = std::fs::metadata(&path).unwrap().len() as i32;  resp.set\_status(response::Status::Ok);  resp.resource = Some(response::Resource::File(file));  resp  } |

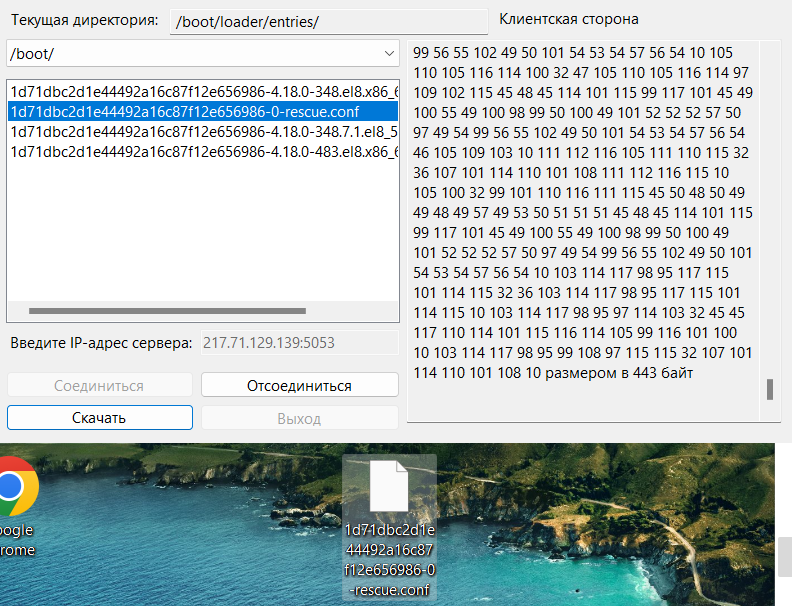
|  |
| --- |
| **Клиент** |
| namespace ClientServer  {  // <auto-generated>  // This file was generated by a tool; you should avoid making direct changes.  // Consider using 'partial classes' to extend these types  // Input: my.proto  // </auto-generated>  #region Designer generated code  #pragma warning disable CS0612, CS0618, CS1591, CS3021, IDE0079, IDE1006, RCS1036, RCS1057, RCS1085, RCS1192  namespace TcpEntities  {  [global::ProtoBuf.ProtoContract()]  public partial class Directory : global::ProtoBuf.IExtensible  {  private global::ProtoBuf.IExtension \_\_pbn\_\_extensionData;  global::ProtoBuf.IExtension global::ProtoBuf.IExtensible.GetExtensionObject(bool createIfMissing)  => global::ProtoBuf.Extensible.GetExtensionObject(ref \_\_pbn\_\_extensionData, createIfMissing);  [global::ProtoBuf.ProtoMember(1, Name = @"path")]  [global::System.ComponentModel.DefaultValue("")]  public string Path { get; set; } = "";  [global::ProtoBuf.ProtoMember(2, Name = @"files")]  public global::System.Collections.Generic.List<string> Files { get; } = new global::System.Collections.Generic.List<string>();  [global::ProtoBuf.ProtoMember(3, Name = @"subdirectories")]  public global::System.Collections.Generic.List<string> Subdirectories { get; } = new global::System.Collections.Generic.List<string>();  }  [global::ProtoBuf.ProtoContract()]  public partial class File : global::ProtoBuf.IExtensible  {  private global::ProtoBuf.IExtension \_\_pbn\_\_extensionData;  global::ProtoBuf.IExtension global::ProtoBuf.IExtensible.GetExtensionObject(bool createIfMissing)  => global::ProtoBuf.Extensible.GetExtensionObject(ref \_\_pbn\_\_extensionData, createIfMissing);  [global::ProtoBuf.ProtoMember(1, Name = @"name")]  [global::System.ComponentModel.DefaultValue("")]  public string Name { get; set; } = "";  [global::ProtoBuf.ProtoMember(2, Name = @"content")]  public byte[] Content { get; set; }  }  [global::ProtoBuf.ProtoContract()]  public partial class Request : global::ProtoBuf.IExtensible  {  private global::ProtoBuf.IExtension \_\_pbn\_\_extensionData;  global::ProtoBuf.IExtension global::ProtoBuf.IExtensible.GetExtensionObject(bool createIfMissing)  => global::ProtoBuf.Extensible.GetExtensionObject(ref \_\_pbn\_\_extensionData, createIfMissing);  [global::ProtoBuf.ProtoMember(1)]  public ActionType action\_type { get; set; }  [global::ProtoBuf.ProtoMember(2, Name = @"path")]  [global::System.ComponentModel.DefaultValue("")]  public string Path { get; set; } = "";  [global::ProtoBuf.ProtoContract()]  public enum ActionType  {  [global::ProtoBuf.ProtoEnum(Name = @"LIST")]  List = 0,  [global::ProtoBuf.ProtoEnum(Name = @"DOWNLOAD")]  Download = 1,  }  }  [global::ProtoBuf.ProtoContract()]  public partial class Response : global::ProtoBuf.IExtensible  {  private global::ProtoBuf.IExtension \_\_pbn\_\_extensionData;  global::ProtoBuf.IExtension global::ProtoBuf.IExtensible.GetExtensionObject(bool createIfMissing)  => global::ProtoBuf.Extensible.GetExtensionObject(ref \_\_pbn\_\_extensionData, createIfMissing);  [global::ProtoBuf.ProtoMember(1)]  public Status status { get; set; }  [global::ProtoBuf.ProtoMember(2, Name = @"directory")]  public Directory Directory  {  get => \_\_pbn\_\_resource.Is(2) ? ((Directory)\_\_pbn\_\_resource.Object) : default;  set => \_\_pbn\_\_resource = new global::ProtoBuf.DiscriminatedUnionObject(2, value);  }  public bool ShouldSerializeDirectory() => \_\_pbn\_\_resource.Is(2);  public void ResetDirectory() => global::ProtoBuf.DiscriminatedUnionObject.Reset(ref \_\_pbn\_\_resource, 2);  private global::ProtoBuf.DiscriminatedUnionObject \_\_pbn\_\_resource;  [global::ProtoBuf.ProtoMember(3, Name = @"file")]  public File File  {  get => \_\_pbn\_\_resource.Is(3) ? ((File)\_\_pbn\_\_resource.Object) : default;  set => \_\_pbn\_\_resource = new global::ProtoBuf.DiscriminatedUnionObject(3, value);  }  public bool ShouldSerializeFile() => \_\_pbn\_\_resource.Is(3);  public void ResetFile() => global::ProtoBuf.DiscriminatedUnionObject.Reset(ref \_\_pbn\_\_resource, 3);  [global::ProtoBuf.ProtoContract()]  public enum Status  {  [global::ProtoBuf.ProtoEnum(Name = @"OK")]  Ok = 0,  [global::ProtoBuf.ProtoEnum(Name = @"NOT\_FOUND")]  NotFound = 1,  [global::ProtoBuf.ProtoEnum(Name = @"PERMISSION\_DENIED")]  PermissionDenied = 2,  [global::ProtoBuf.ProtoEnum(Name = @"ERROR")]  Error = 3,  }  }  } |

**Скриншоты работы программы**

****

****

****

****