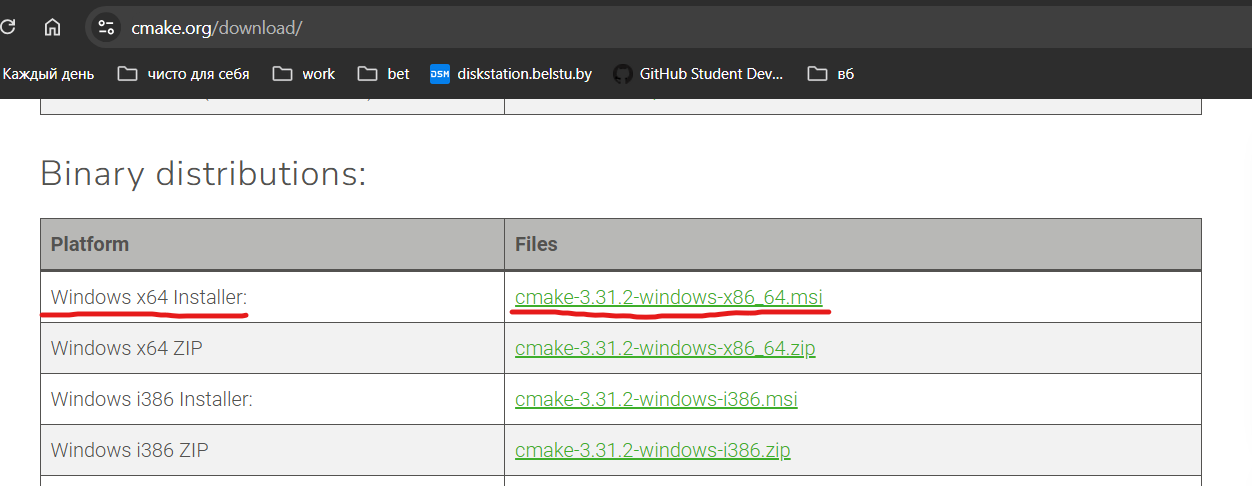
**Руководство по установке gRPC C++ с использованием CMake и vcpkg**

1. Перейдите на официальный сайт CMake: https://cmake.org/download/

2. Выберите версию, подходящую для вашей операционной системы (в нашем случае, выбираем версию для Windows):

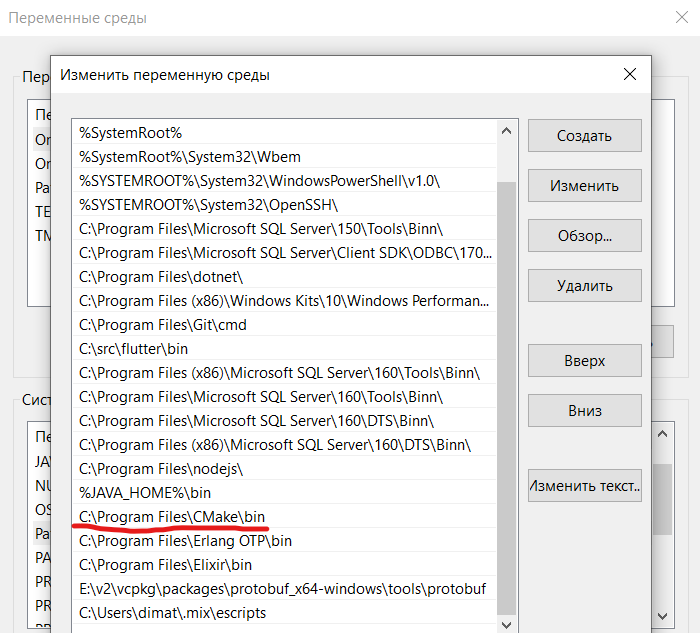
- **Windows**: `cmake-x.x.x-windows-x86\_64.msi`



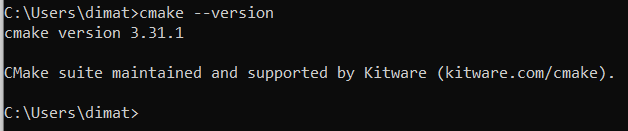
3. Запустите скачанный .msi установщик.

4. Следуйте инструкциям мастера установки:

- Убедитесь, что выбрана опция **Add CMake to the system PATH** для удобного использования CMake из командной строки.

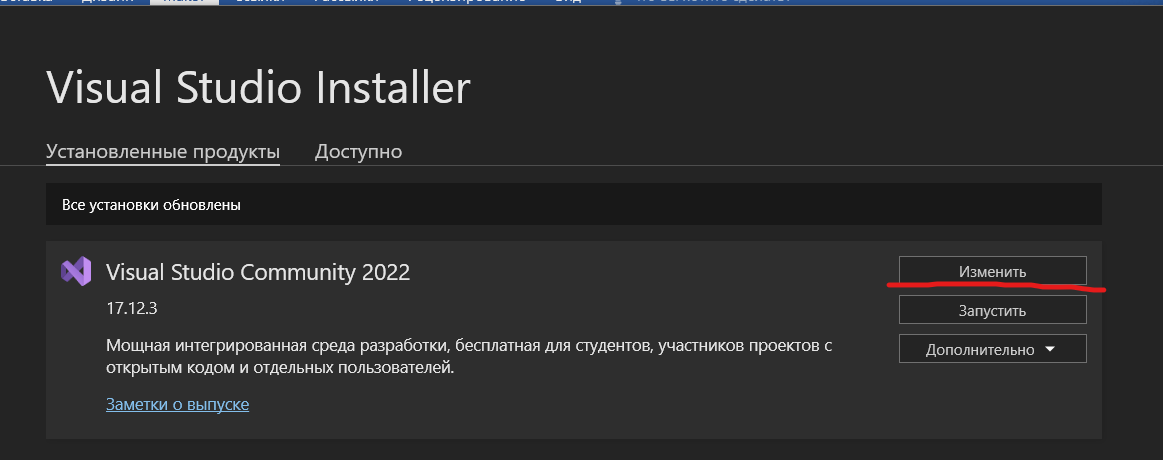


5. После завершения установки откройте командную строку и выполните команду: **cmake --version**

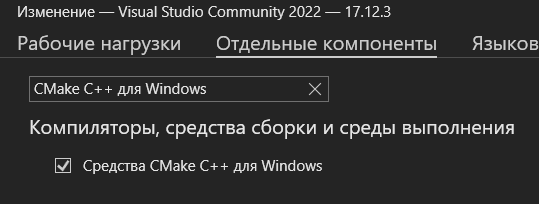


6. Переходим в Пуск, в строке поиска пишем **Visual Studio Installer**, далее запускаем инсталлятор.

7. В окне инсталлятора находим нужную нам версию Visual Studio (если у вас установлены разные версии) и нажимаем на кнопку **Изменить**.



8. В открывшимся окне переходим в раздел **Отдельные компоненты** и в строку поиска пишем **CMake C++ для Windows**.



9. В результатах будет компонент с именем **Средства CMake C++ для Windows**, для установки нажимаем на галочку слева от компонента и соглашаемся с установкой.

10. Создаем папку на диске (к примеру, создадим папку по пути `**E:\v2**`), переходим в нее и делаем клонирование репозитория командой:

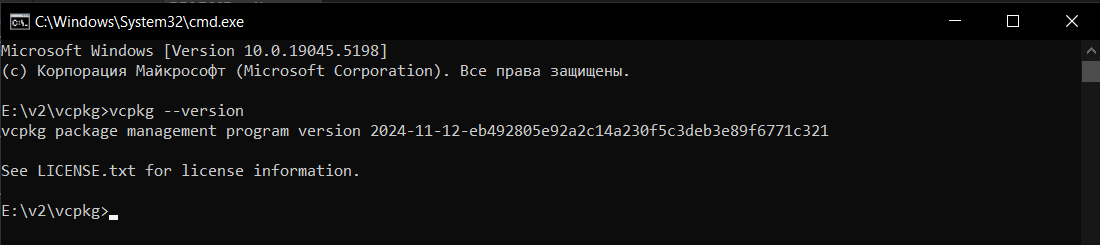
**git clone https://github.com/microsoft/vcpkg.git**

11. Теперь, когда вы клонировали репозиторий vcpkg, перейдите к vcpkg каталогу и выполните скрипт начальной загрузки:

**cd vcpkg и bootstrap-vcpkg.bat**

12. После этого можно проверить корректность установки, для этого пишем:

**vcpkg --version**

****

В случае корректной установки, в консоль будет выведено `vcpkg package management program version 2024-11-12-eb492805e92a2c14a230f5c3deb3e89f6771c321`, где `2024-11-12-eb492805e92a2c14a230f5c3deb3e89f6771c321`- комбинация из даты сборки программы и хэш коммита в репозитории vcpkg.

!!! Важное уточнение !!! После выполнения первой команды ниже, возможно скачается и protobuf, у многих скачивало сразу, выполняем команды по очереди и смотрим, что выводится в консоль.

13. Установите gRPC для архитектуры x64-windows:

**vcpkg install grpc:x64-windows**

14. Установите protobuf (протокол сериализации данных, используемый в gRPC):

**vcpkg install protobuf protobuf:x64-windows**

15. Добавьте поддержку zlib (сжатие данных) для protobuf:

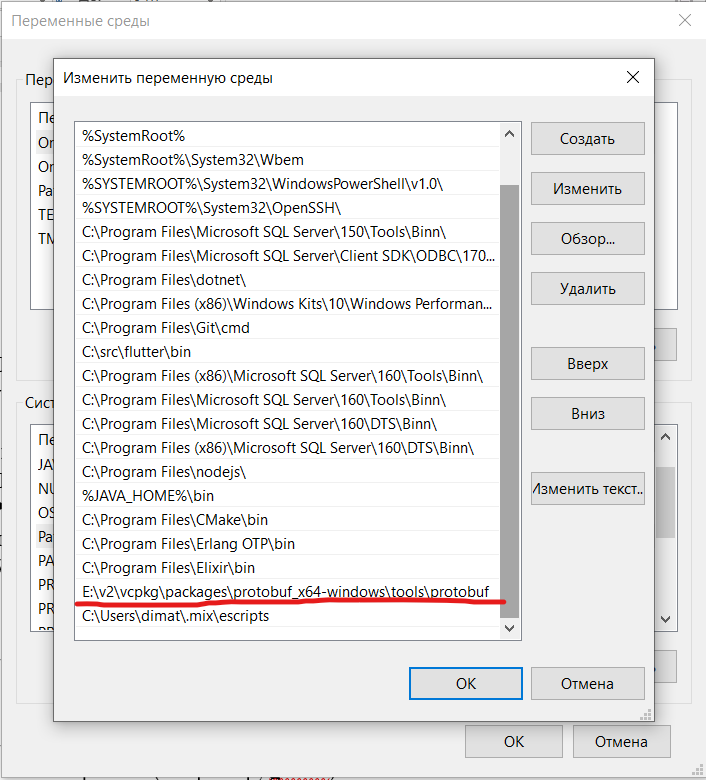
**vcpkg install protobuf[zlib] protobuf[zlib]:x64-windows**

16. Добавьте поддержку zlib (сжатие данных) для protobuf:

**vcpkg integrate install**

После успешной интеграции vcpkg автоматически будет находить библиотеки, установленные через него, и добавлять их в проекты CMake.

17. Нажимаем комбинацию клавиш **Win+S**, в строке поиска пишем **Изменение системных переменных среды**, нажимаем Enter. Переходим в раздел **Переменные среды**, в окне **Системные переменные** находим переменную **PATH**, нажимаем на нее и выбираем **Изменить**. В появившимся окне выбираем пункт и вставляем путь к исполняемому файлу protoc.exe (в моем случае, путь будет вот таким: `E:\v2\vcpkg\packages\protobuf\_x64-windows\tools\protobuf`).



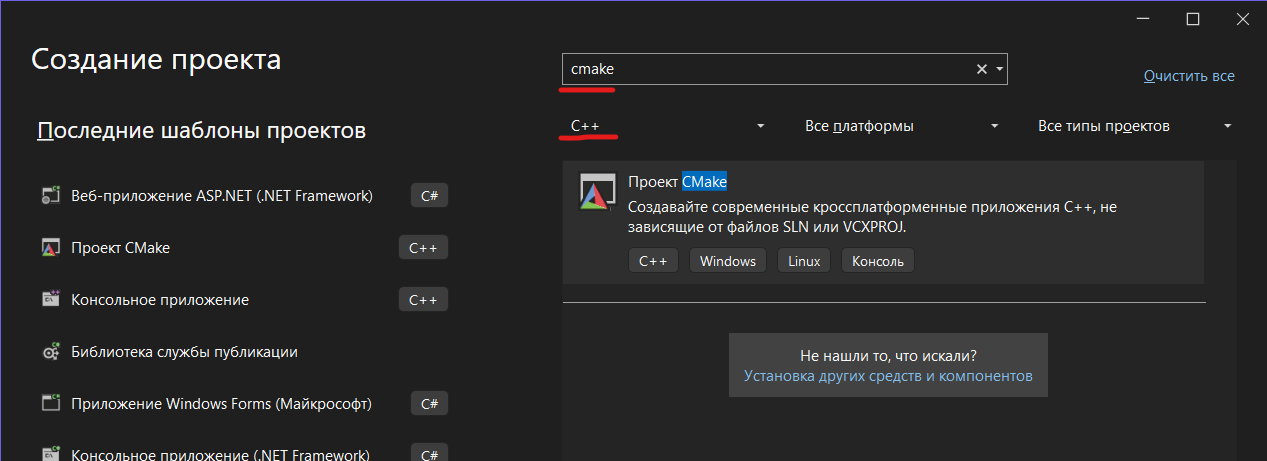
18. Откройте **Visual Studio**.

19. Выберите пункт меню **Создание проекта**.

20. В появившемся окне создания нового проекта выполните следующие шаги:

- Найдите и выберите шаблон **Проект CMake**.

- Нажмите **Далее**.



21. Укажите:

- Имя проекта (например, **gRPC**).

- Путь для сохранения проекта.

- Нажмите **Создать**.

22. В корне проекта создаем папку **proto**.

23. В этой папке создаем файл, к примеру, **math\_operations.proto**.

24. Пример файла **math\_operations.proto**:

|  |
| --- |
| syntax = "proto3";  package math;  service MathService {  rpc Add (Parm2Request) returns (Parm2Result);  rpc Sub (Parm2Request) returns (Parm2Result);  rpc Mul (Parm2Request) returns (Parm2Result);  rpc Div (Parm2Request) returns (Parm2Result);  rpc Pow (Parm2Request) returns (Parm2Result);  }  message Parm2Request {  double x = 1;  double y = 2;  }  message Parm2Result {  double z = 1;  string error = 2;  } |

25. В корневой папке проекта открываем консоль и пишем туда команды:

**protoc -I=proto --cpp\_out=proto proto/math\_operations.proto**

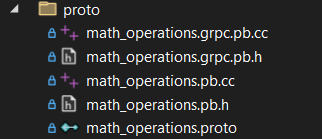
Результат: Создается файл math\_operations.pb.h и math\_operations.pb.cc. Эти файлы содержат:

* Классы для всех сообщений, определенных в math\_operations.proto.
* Методы для сериализации (в двоичный формат) и десериализации.

**protoc -I=proto --grpc\_out=proto --plugin=protoc-gen-grpc="E:\v2\vcpkg\packages\grpc\_x64-windows\tools\grpc\grpc\_cpp\_plugin.exe" proto/math\_operations.proto**

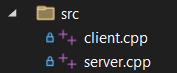
Результат: Создается файл math\_operations.grpc.pb.h и math\_operations.grpc.pb.cc. Эти файлы содержат:

* Код для клиентской и серверной логики (stub и skeleton) для вызова удаленных процедур (RPC), определенных в .proto файле.
* gRPC интерфейсы, которые можно использовать для создания серверов и клиентов.



26. В корне проекта создаем папку **src**.

27. В этой папке создаем два файла: **server.cpp** и **client.cpp**.



28. Пишем логику сервера и клиента, с учетом нашего proto-файла:

server.cpp:

|  |
| --- |
| #include <grpcpp/grpcpp.h>  #include "math\_operations.grpc.pb.h"  #include <cmath>  #include <iostream>  class MathServiceImpl final : public math::MathService::Service {  public:  grpc::Status Add(grpc::ServerContext\* context, const math::Parm2Request\* request,  math::Parm2Result\* response) override {  response->set\_z(request->x() + request->y());  return grpc::Status::OK;  }  grpc::Status Sub(grpc::ServerContext\* context, const math::Parm2Request\* request,  math::Parm2Result\* response) override {  response->set\_z(request->x() - request->y());  return grpc::Status::OK;  }  grpc::Status Mul(grpc::ServerContext\* context, const math::Parm2Request\* request,  math::Parm2Result\* response) override {  response->set\_z(request->x() \* request->y());  return grpc::Status::OK;  }  grpc::Status Div(grpc::ServerContext\* context, const math::Parm2Request\* request,  math::Parm2Result\* response) override {  if (request->y() == 0) {  response->set\_error("Division by zero is not allowed.");  return grpc::Status(grpc::StatusCode::INVALID\_ARGUMENT, "Division by zero");  }  response->set\_z(request->x() / request->y());  return grpc::Status::OK;  }  grpc::Status Pow(grpc::ServerContext\* context, const math::Parm2Request\* request,  math::Parm2Result\* response) override {  response->set\_z(pow(request->x(), request->y()));  return grpc::Status::OK;  }  };  void RunServer() {  std::string server\_address("0.0.0.0:50051");  MathServiceImpl service;  grpc::ServerBuilder builder;  builder.AddListeningPort(server\_address, grpc::InsecureServerCredentials());  builder.RegisterService(&service);  std::unique\_ptr<grpc::Server> server(builder.BuildAndStart());  std::cout << "Server listening on " << server\_address << std::endl;  server->Wait();  }  int main() {  RunServer();  return 0;  } |

client.cpp

|  |
| --- |
| #include <grpcpp/grpcpp.h>  #include "math\_operations.grpc.pb.h"  #include <iostream>  #include <future>  using grpc::Channel;  using grpc::ClientContext;  using grpc::Status;  using math::MathService;  using math::Parm2Request;  using math::Parm2Result;  class MathClient {  public:  MathClient(std::shared\_ptr<Channel> channel)  : stub\_(MathService::NewStub(channel)) {  }  void Add(double x, double y) {  Parm2Request request;  request.set\_x(x);  request.set\_y(y);  Parm2Result response;  ClientContext context;  Status status = stub\_->Add(&context, request, &response);  if (status.ok()) {  std::cout << "Addition result: " << response.z() << std::endl;  } else {  std::cout << "Error: " << response.error() << std::endl;  }  }  void Subtract(double x, double y) {  Parm2Request request;  request.set\_x(x);  request.set\_y(y);  Parm2Result response;  ClientContext context;  Status status = stub\_->Sub(&context, request, &response);  if (status.ok()) {  std::cout << "Subtraction result: " << response.z() << std::endl;  }  else {  std::cout << "Error: " << response.error() << std::endl;  }  }  void Multiply(double x, double y) {  Parm2Request request;  request.set\_x(x);  request.set\_y(y);  Parm2Result response;  ClientContext context;  Status status = stub\_->Mul(&context, request, &response);  if (status.ok()) {  std::cout << "Multiplication result: " << response.z() << std::endl;  }  else {  std::cout << "Error: " << response.error() << std::endl;  }  }  void Divide(double x, double y) {  Parm2Request request;  request.set\_x(x);  request.set\_y(y);  Parm2Result response;  ClientContext context;  Status status = stub\_->Div(&context, request, &response);  if (status.ok()) {  std::cout << "Division result: " << response.z() << std::endl;  }  else {  std::cout << "Error: " << status.error\_message() << std::endl;  }  }  void Power(double x, double y) {  Parm2Request request;  request.set\_x(x);  request.set\_y(y);  Parm2Result response;  ClientContext context;  Status status = stub\_->Pow(&context, request, &response);  if (status.ok()) {  std::cout << "Power result: " << response.z() << std::endl;  }  else {  std::cout << "Error: " << response.error() << std::endl;  }  }  std::future<void> MultiplyAsync(double x, double y) {  return std::async(std::launch::async, [this, x, y]() {  Parm2Request request;  request.set\_x(x);  request.set\_y(y);  Parm2Result response;  ClientContext context;  Status status = stub\_->Mul(&context, request, &response);  if (status.ok()) {  std::cout << "Multiplication result (async): " << response.z() << std::endl;  }  else {  std::cout << "Error: " << response.error() << std::endl;  }  });  }  private:  std::unique\_ptr<MathService::Stub> stub\_;  };  int main() {  MathClient client(grpc::CreateChannel("localhost:50051", grpc::InsecureChannelCredentials()));  client.Add(3, 4);  client.Subtract(10, 5);  client.Multiply(6, 7);  auto multiplyFuture = client.MultiplyAsync(6, 7);  multiplyFuture.get();  client.Divide(10, 2);  client.Divide(10, 0);  client.Power(2, 3);  std::cout << "Press Enter to exit...";  std::cin.get();  return 0;  } |

29. В корневой директории проекта Visual Studio создаст файл **CMakeLists.txt**.

Откройте его и добавьте или отредактируйте следующие строки:

|  |
| --- |
| cmake\_minimum\_required(VERSION 3.15)  project(gRPC)  set(re2\_DIR "E:/v2/vcpkg/installed/x64-windows/share/re2")  set(absl\_DIR "E:/v2/vcpkg/installed/x64-windows/share/absl")  set(gRPC\_DIR "E:/v2/vcpkg/installed/x64-windows/share/grpc")  set(utf8\_range\_DIR "E:/v2/vcpkg/installed/x64-windows/share/utf8\_range")  set(Protobuf\_DIR "E:/v2/vcpkg/installed/x64-windows/share/protobuf")  set(ZLIB\_INCLUDE\_DIR "E:/v2/vcpkg/installed/x64-windows/include")  set(ZLIB\_LIBRARY "E:/v2/vcpkg/installed/x64-windows/lib/zlib.lib")  set(c-ares\_DIR "E:/v2/vcpkg/installed/x64-windows/share/c-ares")  set(OPENSSL\_ROOT\_DIR "E:/v2/vcpkg/installed/x64-windows")  find\_package(gRPC CONFIG REQUIRED)  find\_package(Protobuf CONFIG REQUIRED)  find\_package(OpenSSL REQUIRED)  find\_package(c-ares CONFIG REQUIRED)  if(NOT TARGET ZLIB::ZLIB)  find\_path(ZLIB\_INCLUDE\_DIR NAMES zlib.h PATHS ${ZLIB\_INCLUDE\_DIR})  find\_library(ZLIB\_LIBRARY NAMES zlib PATHS ${ZLIB\_LIBRARY})  add\_library(ZLIB::ZLIB STATIC IMPORTED)  set\_target\_properties(ZLIB::ZLIB PROPERTIES  INTERFACE\_INCLUDE\_DIRECTORIES ${ZLIB\_INCLUDE\_DIR}  INTERFACE\_LINK\_LIBRARIES ${ZLIB\_LIBRARY}  )  endif()  include\_directories(proto)  add\_executable(server src/server.cpp proto/math\_operations.grpc.pb.cc proto/math\_operations.pb.cc)  add\_executable(client src/client.cpp proto/math\_operations.grpc.pb.cc proto/math\_operations.pb.cc)  target\_link\_libraries(server PRIVATE gRPC::grpc++ gRPC::grpc++\_reflection protobuf::libprotobuf ZLIB::ZLIB OpenSSL::SSL OpenSSL::Crypto)  target\_link\_libraries(client PRIVATE gRPC::grpc++ gRPC::grpc++\_reflection protobuf::libprotobuf ZLIB::ZLIB OpenSSL::SSL OpenSSL::Crypto) |

30. В корне проекта создаем папку **build**.

31. Переходим в созданную папку, открываем консоль и прописываем следующие команды:

**cmake .. -DCMAKE\_TOOLCHAIN\_FILE=E:/v2/vcpkg/scripts/buildsystems/vcpkg.cmake**

**cmake --build . --config Release**

Первая команда настраивает проект и готовит сборочную систему, интегрируя зависимости через vcpkg.

Вторая команда компилирует проект в указанной конфигурации (Release), создавая итоговые .exe файлы.

32. Переходим в папку **Release**, которая создалась в папке **build** и видим два файла - **server.exe** и **client.exe**. Запускаем вначале сервер, потом клиент.

