10- Практическое задание — Разработка распределенную систему управления для симуляции работы группы дронов (Многопоточность, сокеты и базы данных)

Table of Contents

Цель

Цель данной лабораторной работы заключается в разработке и реализации распределенной системы управления группой дронов с использованием принципов параллельной обработки, программирования сокетов и работы с базой данных в C++ для симуляции работы команды дронов по выполнению задачи обследования зданий под углом 360 градусов.

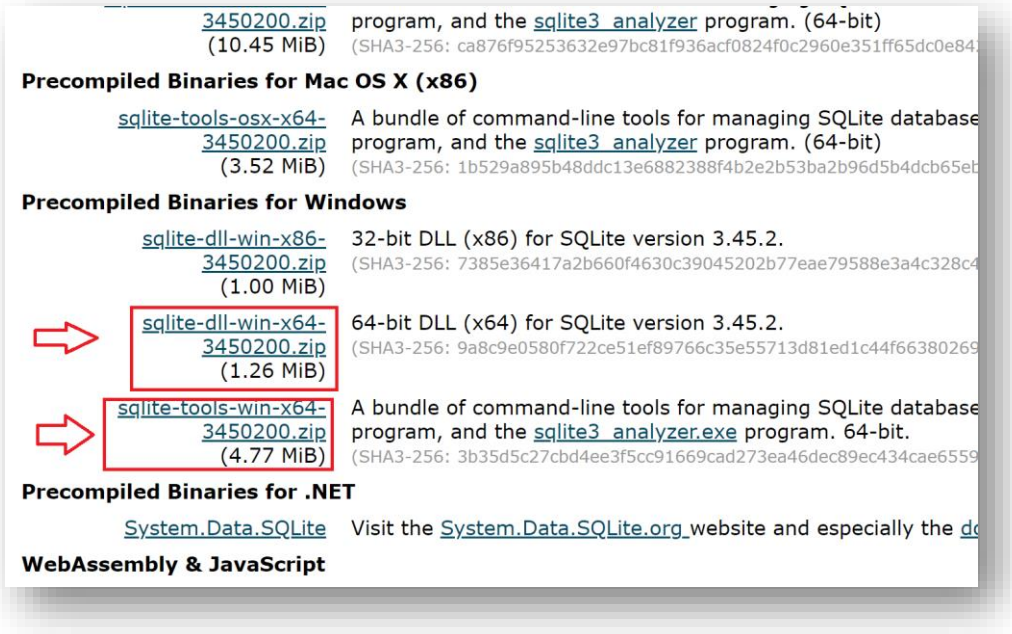
1. Подключение к базе данных в C++

1. Перейдите на https://www.sqlite.org/download.html и скачайте последнюю версию следующих файлов:

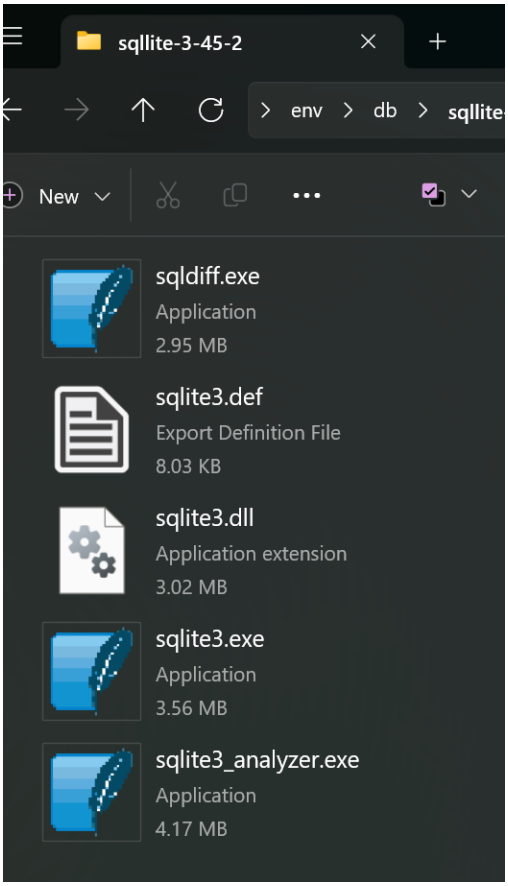
sqlite-dll-win

sqlite-tools-win

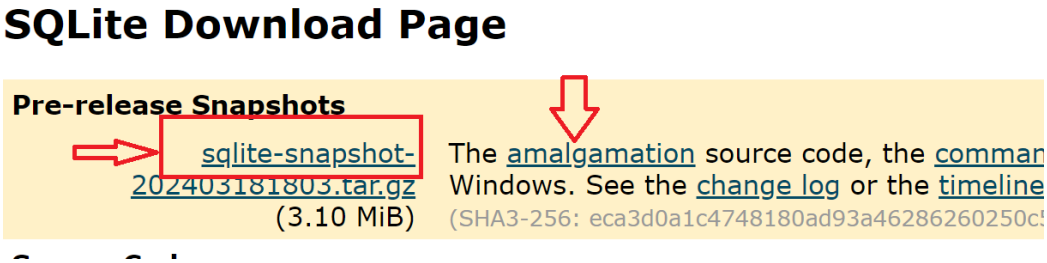
Например, если ваша операционная система Windows 64-битная:



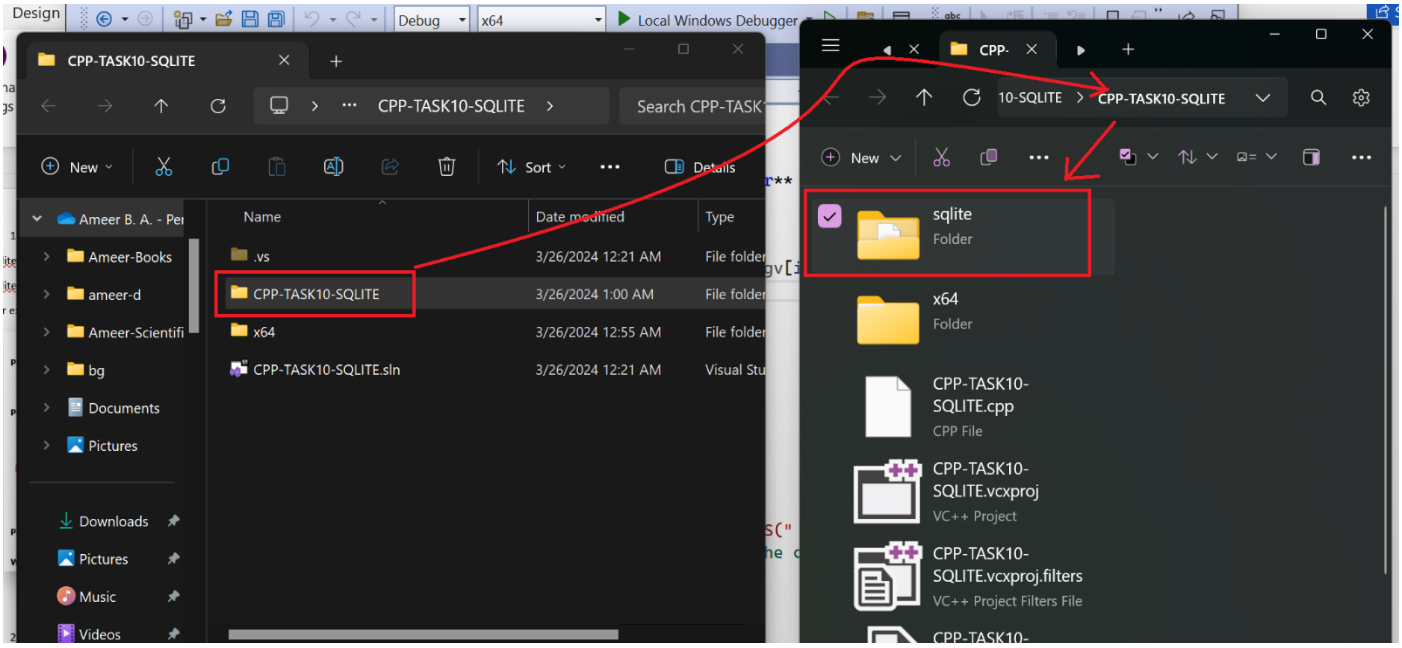
2. Create a folder C:\>sqlite and unzip above two zipped files in this folder. So, the results will be similar to the following:



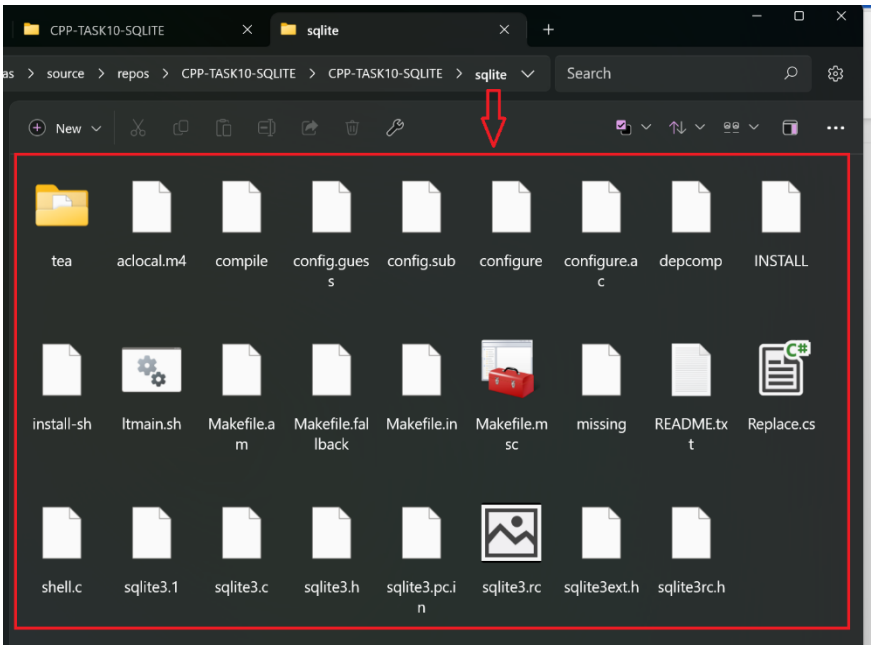
3. Добавьте C:>sqlite в переменную среды PATH и перезагрузите ваш компьютер.
4. Зайдите снова на https://www.sqlite.org/download.html и скачайте последнюю версию следующего файла, который содержит (исходный код амальгамации), например:



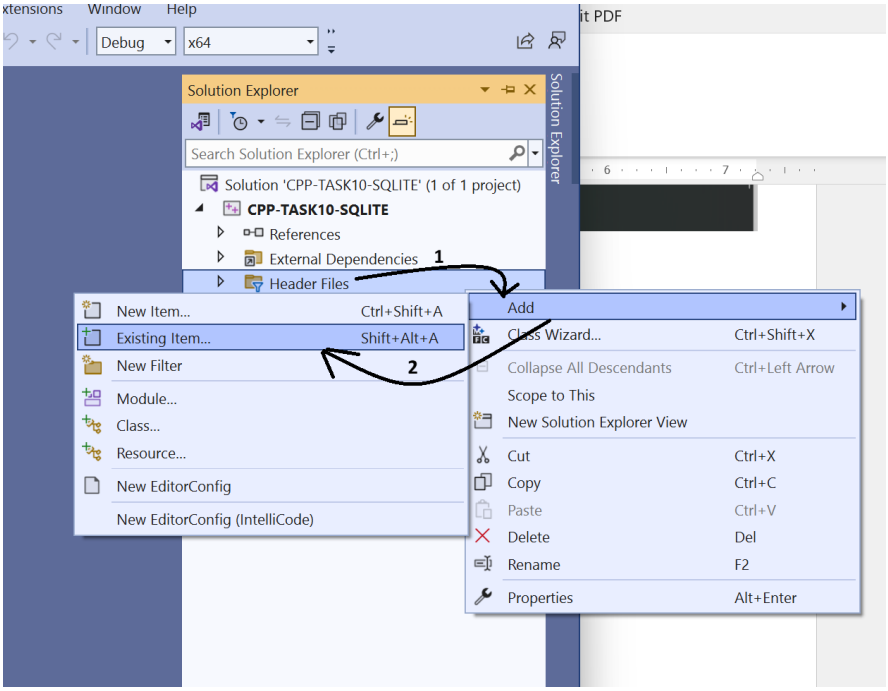
5. В папке вашего решения Visual Studio откройте проектную папку, создайте папку с именем "sqlite", распакуйте все содержимое скачанного файла (указанного в пункте 4 выше) и скопируйте всё его содержимое во вновь созданную папку "sqlite":



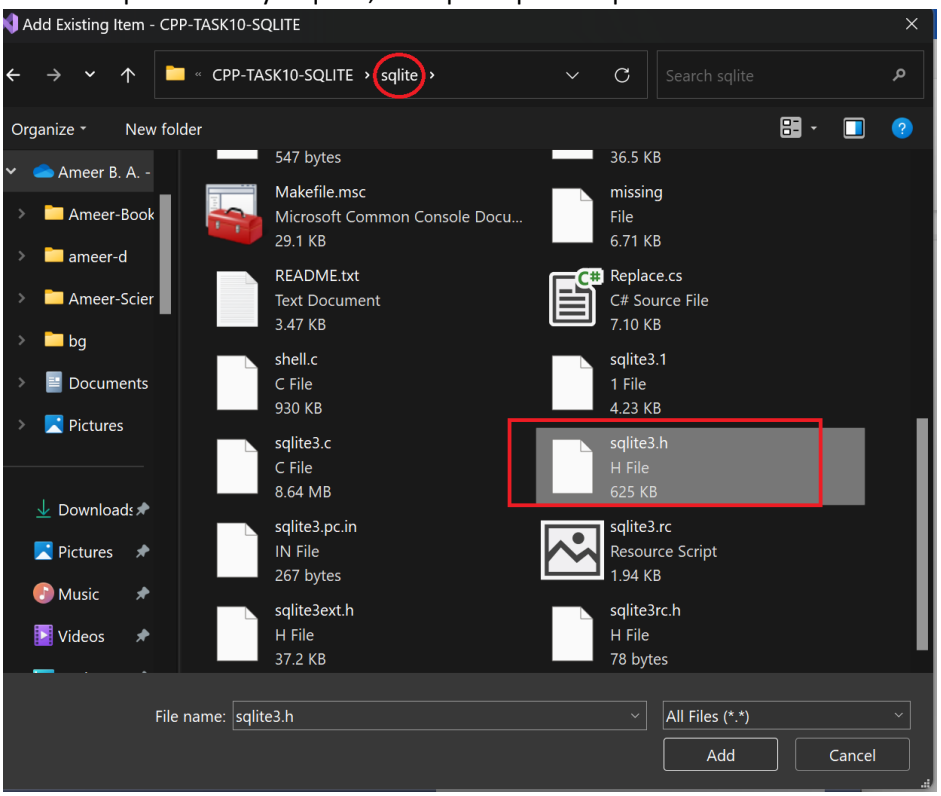
Таким образом, содержимое папки "sqlite" должно быть похоже на следующее:



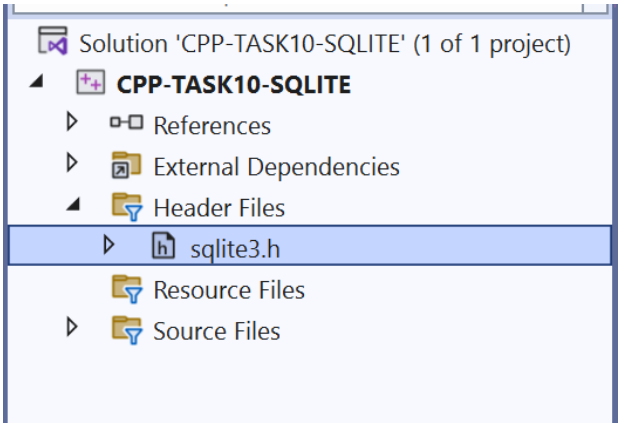
6. Теперь в окне "Обозреватель решений" щёлкните правой кнопкой мыши по папке, которая называется "Header Files", выберите "Добавить", а затем "Существующий элемент..." как показано ниже:



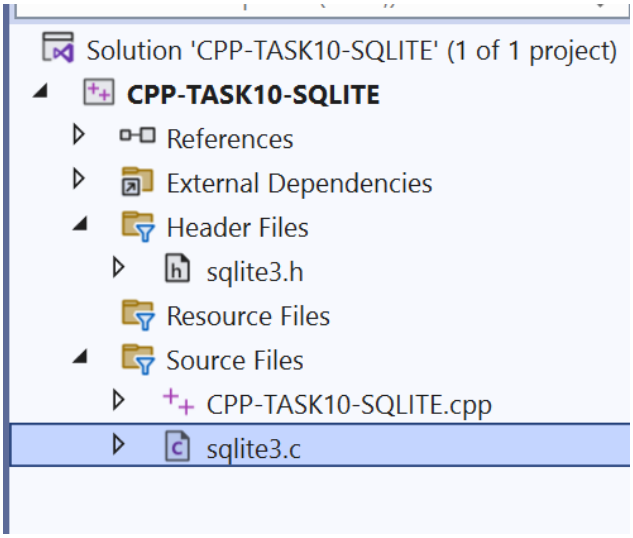
Затем откройте папку "sqlite", выберите файл "sqlite3.h" и нажмите на кнопку "Добавить" внизу



Вы увидите, что файл "sqlite3.h" был добавлен в ваш проект в папку "Header Files" следующим образом:



7. Повторите то же самое с папкой, называемой "Source Files" в вашем проекте, но на этот раз вместо sqlite3.h вам нужно добавить файл "sqlite3.c" в папку "Source Files" вашего проекта, и результат будет следующим:



8. Теперь вы готовы реализовать первый пример "hello world" для базы данных на C++ с использованием SQLite. Код следующий:

```cpp
// Подключение библиотеки для работы с потоками ввода/вывода.
#include <iostream>
// Подключение библиотеки SQLite для работы с базой данных SQLite.
#include "sqlite/sqlite3.h"

// Объявление статической функции обратного вызова, используемой библиотекой SQLite.
// data - пользовательские данные, передаваемые в функцию.
// argc - количество столбцов в результате.
// argv - значения данных в строке результата.
// azColName - имена столбцов в результате.
static int callback(void* data, int argc, char** argv, char** azColName) {
    int i;
    // Вывод пользовательских данных, переданных в функцию.
    std::cout << (const char*)data << std::endl;
    // Цикл для обработки каждой колонки в результате запроса.
    for (i = 0; i < argc; i++) {
        // Вывод имени и значения каждого столбца. Если значение NULL, выводится "NULL".
        std::cout << azColName[i] << ": " << (argv[i] ? argv[i] : "NULL") << std::endl;
    }
    // Вывод пустой строки для разделения результатов.
    std::cout << "\n";
    // Возврат 0 означает успешное выполнение функции.
    return 0;
}
```

```
// Основная функция программы.
int main() {
    // Объявление указателя на структуру базы данных SQLite.
    sqlite3* DB;
    // Указатель для хранения сообщения об ошибке от функций SQLite.
    char* messageError;
    // Открытие или создание базы данных с именем example1.db.
    int exit = sqlite3_open("example1.db", &DB);
    // Строка для передачи в функцию обратного вызова.
    std::string data("CALLBACK FUNCTION");

    // SQL запрос для создания таблицы, вставки данных и выборки данных.
    std::string sql = "CREATE TABLE IF NOT EXISTS GREETINGS("
        "ID INTEGER PRIMARY KEY AUTOINCREMENT, " // Идентификатор записи, автоматически увеличивающийся.
        "MESSAGE TEXT NOT NULL);" // Столбец для хранения сообщений.
        "INSERT INTO GREETINGS (MESSAGE) " // Вставка строки с сообщением.
        "VALUES ('Hello, World');" // Значение вставляемого сообщения.
        "SELECT * FROM GREETINGS;"; // Выборка всех записей из таблицы.

    // Выполнение SQL запроса. При ошибке выводится сообщение.
    exit = sqlite3_exec(DB, sql.c_str(), callback, (void*)data.c_str(), &messageError);
    if (exit != SQLITE_OK) {
        // Вывод сообщения об ошибке, если выполнение SQL не удалось.
        std::cerr << "Error in executing SQL: " << messageError << std::endl;
        // Освобождение памяти, выделенной под сообщение об ошибке.
        sqlite3_free(messageError);
    }
    else {
        // Вывод сообщения об успешном выполнении операций.
        std::cout << "Records created and selected successfully" << std::endl;
    }

    // Закрытие соединения с базой данных.
    sqlite3_close(DB);
    // Возврат 0 из main() означает успешное завершение программы.
    return 0;
}
```

.

2. Самостоятельная задача: Разработка распределённой системы управления группой дронов

Давайте представим команду дронов с возможностью самоорганизации. Предположим, что миссия команды дронов заключается в записи видео одного из зданий под углом 360 градусов. То есть, если команда состоит из 3 дронов, каждый дрон займет место, позволяющее ему охватить 120 градусов здания. Если один из дронов выходит из строя, оставшиеся два дрона изменят свое местоположение так, чтобы каждый охватывал 180 градусов здания. Или если добавляется четвертый дрон, все дроны займут позиции, позволяющие каждому охватывать по 90 градусов здания, и так далее для всех остальных возможностей.

Теперь ваша задача - создать симуляцию этой идеи, где:

1. Каждый дрон является независимой программой (клиентом), которая получает и отправляет данные серверу по протоколу TCP/IP.
2. Данные, отправляемые клиентом на сервер, представляют собой только его идентификатор, прикрепленный к тому, только что он вступил в службу (только что был запущен) или выйдет из службы (будет остановлен), на основе инструкций пользователя.
3. Несколько клиентов работают одновременно, и вы пытаетесь закрыть выполнение одного из них или запустить другого.
4. Сервер, на основе сообщений, полученных от клиентов, выполняет следующее:
    A. Он сохраняет каждое сообщение, поступающее от клиента, в базе данных в отдельной таблице.
    B. На основе полученной от клиентов информации сервер рассчитывает, сколько клиентов в данный момент действительно находятся на службе, и на основе этого делит угол в 360 градусов между ними и отправляет сообщение всем клиентам, содержащее долю каждого из них в угле. Например, если есть 4 клиента, сервер отправит сообщение, содержащее число 90, как выражение угла в 90 градусов, который каждый клиент хочет охватить. Сервер также сохраняет это сообщение в отдельной таблице в базе данных.
    C. Когда каждый клиент получает сообщение от сервера, содержащее желаемый угол, например, 90, клиент будет выводить его в консоль.
    D. Вы должны попытаться создать несколько клиентов одновременно, затем выключить одного из них или добавить другого и убедиться, что этот процесс корректен, как описано выше.
5. Вы должны использовать параллельную обработку, чтобы сервер мог обслуживать более одного клиента одновременно. Как упоминалось в задаче, вы должны использовать базу данных, а не простой файл. Вы можете использовать любую систему управления базами данных по своему усмотрению. Также от вас требуется использовать программирование сокетов для управления коммуникацией между сервером и клиентами.