Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Программирование сетевых приложений

Студент: Немкович А.В.

ФИТ 3 курс 1 группа

Преподаватель: Некрасова А.П.

Минск 2024

**Выполнение индивидуального задания на основе программ реализованных для лабораторных работ №8-10**

В качестве индивидуального задания было выбрано следующее: Администратор оправляет письмо серверу, в котором содержится расписание загрузки библиотек.

Ниже будет представлено описание и пример кода, с уже внедренной дополнительной функциональностью. А именно будет описание изменений в программах ConcurentServer и RConsole(администратор).

RConsole.cpp (main)

|  |
| --- |
| cout << "\n -------------------Команды---------------- \n";  cout << "1 - Start\tразрешить подключение клиентов к серверу\n";  cout << "2 - Stop\tзапретить подключение клиентов к серверу\n";  cout << "3 - Exit\tзавершить работу сервера\n";  cout << "4 - Statistics\tвывод статистики\n";  cout << "5 - Wait\tприостановить подключение клиентов, пока не обслужится последний клиент, подключенный к серверу\n";  cout << "6 - Shutdown\tравносильна последовательности команд: wait, exit\n";  cout << "7 - Отправить расписание через Mailslot\n";  cout << "0 - Выход из консоли\n";  cout << "\n ----------------------------------------- \n"; |

Добавлена опция с командой 7, которая инициирует отправку расписания через Mailslot. Эта команда позволяет администратору сервера содержимое текстового файла с расписанием.

ConcurenServer.h

|  |
| --- |
| enum TalkersCommand {  START,  STOP,  EXIT,  WAIT,  SHUTDOWN,  GETCOMMAND,  START\_MAILSLOT  }; |

В ConcurenServer.h было добавлено новое значение в перечисление команд TalkersCommand для поддержки новой команды отправки расписания.

RConsole.cpp (main)

|  |
| --- |
| if (command > 0 && command <= 7) {  if (command == 7) {  string scheduleFileName = "scheduleForServer.txt"; // имя файла с расписанием  ifstream file(scheduleFileName);  if (!file.is\_open()) {  cerr << "Ошибка: Невозможно открыть файл " << scheduleFileName << endl;  continue;  }  // содержимое файла  stringstream buffer;  buffer << file.rdbuf();  string fileContent = buffer.str();  file.close();  // отправляем через Mailslot  HANDLE hMailslot = CreateFile(L"\\\\.\\mailslot\\ServerMailslot",  GENERIC\_WRITE,  FILE\_SHARE\_READ,  NULL,  OPEN\_EXISTING,  FILE\_ATTRIBUTE\_NORMAL,  NULL);  if (hMailslot == INVALID\_HANDLE\_VALUE) {  cerr << "Ошибка: Невозможно подключиться к Mailslot" << endl;  continue;  }  DWORD bytesWritten;  if (!WriteFile(hMailslot, fileContent.c\_str(), fileContent.size() + 1, &bytesWritten, NULL)) {  cerr << "Ошибка записи в Mailslot" << endl;  }  else {  cout << "Файл " << scheduleFileName << " успешно отправлен через Mailslot" << endl;  }  CloseHandle(hMailslot);  } |

Этот фрагмент кода реализует отправку содержимого файла с расписанием через Mailslot. Проверка на введенную команду и если значение равно 7, выполняется дальнейшая обработка.

Открывается файл с именем "scheduleForServer.txt" для чтения. Если файл не удается открыть, выводится сообщение об ошибке. Если файл открыт успешно, его содержимое считывается в строку с использованием stringstream. После этого файл закрывается.

Создается подключение к Mailslot с именем "\\.\mailslot\ServerMailslot". Если подключение к Mailslot прошло успешно, содержимое файла отправляется через Mailslot с помощью функции WriteFile. В случае успешной записи выводится сообщение об успешной отправке файла, в противном случае — ошибка.

После отправки данных соединение с Mailslot закрывается с помощью CloseHandle.

ConcurenServer (ScheduleEntry)

|  |
| --- |
| struct ScheduleEntry {  SYSTEMTIME time;  std::string libraryName;  // Для сортировки по времени  bool operator<(const ScheduleEntry& other) const {  if (time.wYear != other.time.wYear) return time.wYear < other.time.wYear;  if (time.wMonth != other.time.wMonth) return time.wMonth < other.time.wMonth;  if (time.wDay != other.time.wDay) return time.wDay < other.time.wDay;  if (time.wHour != other.time.wHour) return time.wHour < other.time.wHour;  return time.wMinute < other.time.wMinute;  }  }; |

Эта структура используется для хранения информации о записи в расписании. Каждая запись включает в себя:

* time: время, в которое должна быть загружена библиотека.
* libraryName: имя библиотеки, которую нужно загрузить в указанное время.

Переопределила оператор < для того, чтобы можно было сортировать записи по времени (год, месяц, день, час, минута).

ConcurenServer (ParseScheduleLine)

|  |
| --- |
| bool ParseScheduleLine(const std::string& line, ScheduleEntry& entry) {  std::istringstream lineStream(line);  std::string date, time, libraryName;  // извлечение данных  std::getline(lineStream, date, ' ');  std::getline(lineStream, time, ' ');  std::getline(lineStream, libraryName);  int year, month, day;  if (sscanf\_s(date.c\_str(), "%d-%d-%d", &year, &month, &day) != 3) {  return false;  }  int hour, minute;  if (sscanf\_s(time.c\_str(), "%d:%d", &hour, &minute) != 2) {  return false;  }  // получаем текущую дату  SYSTEMTIME st;  GetLocalTime(&st);  entry.time = st;  entry.time.wYear = year;  entry.time.wMonth = month;  entry.time.wDay = day;  entry.time.wHour = hour;  entry.time.wMinute = minute;  entry.libraryName = libraryName;  return true;  } |

Функция разбирает строку из расписания и заполняет структуру ScheduleEntry. Строка должна содержать дату, время и имя библиотеки в формате: YYYY-MM-DD HH:MM LibraryName

Если строка не соответствует этому формату, функция возвращает false. В противном случае она заполняет структуру ScheduleEntry и возвращает true.

ConcurenServer (IsTimeInPast)

|  |
| --- |
| bool IsTimeInPast(const SYSTEMTIME& time) {  SYSTEMTIME currentTime;  GetLocalTime(&currentTime);  if (currentTime.wYear > time.wYear) return true;  if (currentTime.wMonth > time.wMonth) return true;  if (currentTime.wDay > time.wDay) return true;  if (currentTime.wHour > time.wHour) return true;  if (currentTime.wMinute > time.wMinute) return true;  return false;  } |

Эта функция проверяет, не прошло ли заданное время относительно текущего времени. Если текущее время больше, чем заданное, функция возвращает true. Иначе возвращает false.

ConcurenServer (LoadLibraryAtTime)

|  |
| --- |
| void LoadLibraryAtTime(const ScheduleEntry& entry) {  if (IsTimeInPast(entry.time)) {  std::cout << "Время для загрузки библиотеки " << entry.libraryName << " уже прошло." << std::endl;  return;  }  SYSTEMTIME currentTime;  GetLocalTime(&currentTime);  // ожидания настпуления времени из расписания  while (currentTime.wHour < entry.time.wHour || (currentTime.wHour == entry.time.wHour && currentTime.wMinute < entry.time.wMinute)) {  std::this\_thread::sleep\_for(std::chrono::seconds(1));  GetLocalTime(&currentTime);  }  // загружаем библиотеку  wchar\_t wDllName[MAX\_PATH];  MultiByteToWideChar(CP\_ACP, 0, entry.libraryName.c\_str(), -1, wDllName, MAX\_PATH);  HMODULE st = LoadLibrary(wDllName);  if (st) {  std::cout << "Библиотека " << entry.libraryName << " загружена успешно." << std::endl;  }  else {  std::cerr << "Ошибка загрузки библиотеки " << entry.libraryName << ": " << GetLastError() << std::endl;  }  } |

Функция загружает динамическую библиотеку в указанное время, используя LoadLibrary. Если заданное время уже прошло, выводится сообщение, что время загрузки библиотеки прошло. В противном случае программа будет ожидать наступления нужного времени, проверяя текущие дату и время, и загружает библиотеку, когда время совпадает.

ConcurenServer (MailslotServer)

|  |
| --- |
| DWORD WINAPI MailslotServer(LPVOID pPrm) {  HANDLE hMailslot = CreateMailslot(L"\\\\.\\mailslot\\ServerMailslot", 0, MAILSLOT\_WAIT\_FOREVER, NULL);  if (hMailslot == INVALID\_HANDLE\_VALUE) {  std::cerr << "Ошибка создания Mailslot: " << GetLastError() << std::endl;  return 1;  }  std::cout << "Mailslot создан и ожидает сообщений..." << std::endl;  char buffer[1024];  DWORD bytesRead;  while (true) {  if (ReadFile(hMailslot, buffer, sizeof(buffer) - 1, &bytesRead, NULL)) {  buffer[bytesRead] = '\0';  std::cout << "Получено расписание:\n" << buffer << std::endl;  std::ofstream receivedFile("received\_schedule.txt");  if (receivedFile.is\_open()) {  receivedFile << buffer;  receivedFile.close();  std::cout << "Содержимое сохранено в файл received\_schedule.txt" << std::endl;  }  else {  std::cerr << "Ошибка: Невозможно сохранить содержимое в файл" << std::endl;  }  std::istringstream iss(buffer);  std::string line;  std::vector<ScheduleEntry> scheduleEntries;  while (std::getline(iss, line)) {  if (line.empty()) continue;  ScheduleEntry entry;  if (ParseScheduleLine(line, entry)) {  scheduleEntries.push\_back(entry);  }  else {  std::cerr << "Ошибка при разборе строки расписания: " << line << std::endl;  }  }  std::sort(scheduleEntries.begin(), scheduleEntries.end());  std::cout << "Полученное расписание:" << std::endl;  for (const auto& entry : scheduleEntries) {  std::ostringstream dateStream;  dateStream << (entry.time.wDay < 10 ? "0" : "") << entry.time.wDay << "."  << (entry.time.wMonth < 10 ? "0" : "") << entry.time.wMonth << "."  << entry.time.wYear;  std::cout << "Дата: " << dateStream.str()  << " Время: " << entry.time.wHour << ":" << (entry.time.wMinute < 10 ? "0" : "") << entry.time.wMinute  << " Библиотека: " << entry.libraryName << std::endl;  }  for (const auto& entry : scheduleEntries) {  LoadLibraryAtTime(entry);  }  }  else {  std::cerr << "Ошибка чтения из Mailslot: " << GetLastError() << std::endl;  }  }  DWORD rc = 0;  ExitThread(rc);  CloseHandle(hMailslot);  return 0;  } |

Эта функция запускает сервер для работы с Mailslot, который будет слушать входящие сообщения. Когда сервер получает данные, они обрабатываются и сохраняются в файл. Входящие данные интерпретируются как расписание загрузки библиотек, которое затем парсится и сортируется по времени. Для каждой записи из расписания выполняется функция загрузки библиотеки.

RConsole.cpp (main)

|  |
| --- |
| hMailslotThread = CreateThread(NULL, NULL, MailslotServer, (LPVOID)&cmd, NULL, NULL);  if (hAcceptServer && hDispatchServer && hGarbageCleaner && hConsolePipe && hResponseServer && hMailslotThread) {  WaitForSingleObject(hAcceptServer, INFINITE);  WaitForSingleObject(hDispatchServer, INFINITE);  WaitForSingleObject(hGarbageCleaner, INFINITE);  WaitForSingleObject(hConsolePipe, INFINITE);  WaitForSingleObject(hResponseServer, INFINITE);  WaitForSingleObject(hMailslotThread, INFINITE);  CloseHandle(hAcceptServer);  CloseHandle(hDispatchServer);  CloseHandle(hGarbageCleaner);  CloseHandle(hConsolePipe);  CloseHandle(hResponseServer);  CloseHandle(hMailslotThread);  } |

В главной функции программы создается отдельный поток для обработки Mailslot с использованием функции MailslotServer. Поток будет слушать Mailslot и обрабатывать полученные данные в фоновом режиме.

**Тестирование.**

Для демонстрации рабоспособности были созданы 2 простые библиотеки:

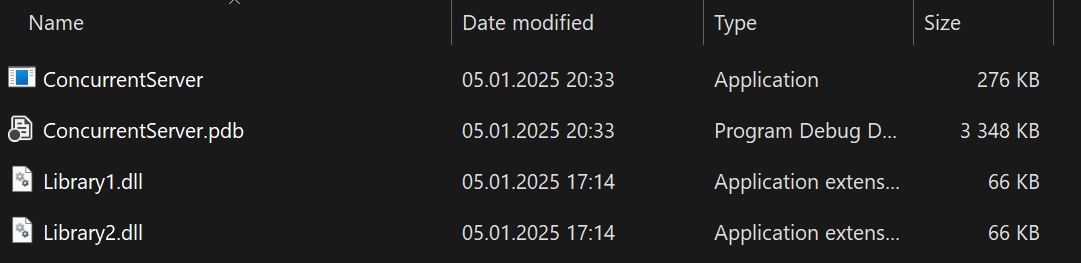
Library1.cpp

|  |
| --- |
| #include "pch.h"  #include <iostream>  #define EXPORT \_\_declspec(dllexport)  BOOL APIENTRY DllMain( HMODULE hModule,  DWORD ul\_reason\_for\_call,  LPVOID lpReserved  )  {  switch (ul\_reason\_for\_call)  {  case DLL\_PROCESS\_ATTACH:  case DLL\_THREAD\_ATTACH:  case DLL\_THREAD\_DETACH:  case DLL\_PROCESS\_DETACH:  break;  }  return TRUE;  }  extern "C" {  EXPORT void HelloWorld() {  std::cout << "Hello from Library1!" << std::endl;  }  } |

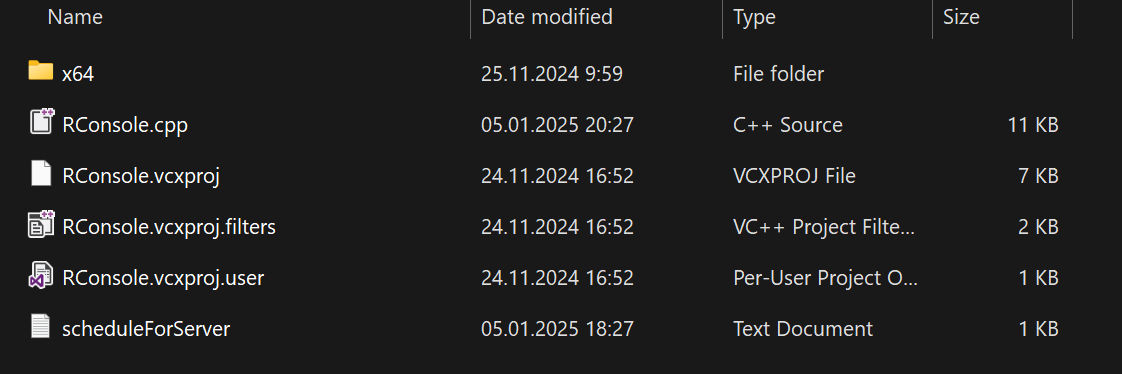
Library2.cpp

|  |
| --- |
| #include "pch.h"  #include <iostream>  #define EXPORT \_\_declspec(dllexport)  BOOL APIENTRY DllMain( HMODULE hModule,  DWORD ul\_reason\_for\_call,  LPVOID lpReserved  )  {  switch (ul\_reason\_for\_call)  {  case DLL\_PROCESS\_ATTACH:  case DLL\_THREAD\_ATTACH:  case DLL\_THREAD\_DETACH:  case DLL\_PROCESS\_DETACH:  break;  }  return TRUE;  }  extern "C" {  EXPORT void Greetings() {  std::cout << "Greetings from Library2!" << std::endl;  }  } |

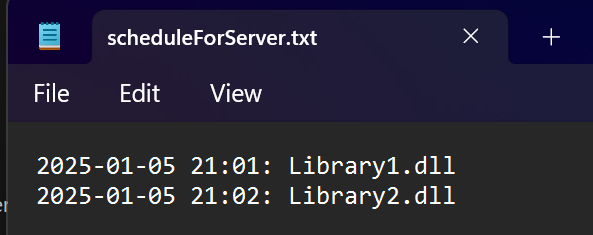
После сборки решений файлы с расширением .dll были размещены в каталоге с исполняемым файлом ConcurenServer.exe.

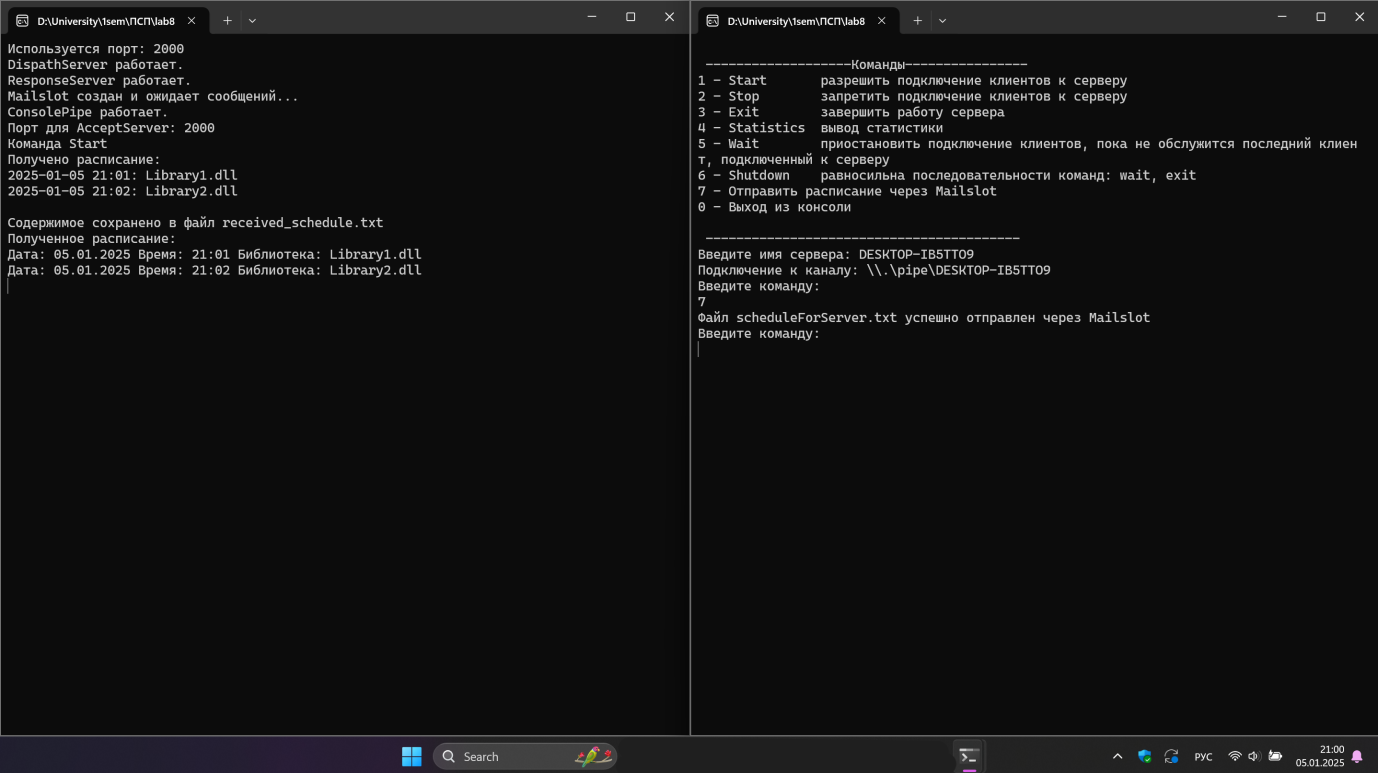


Текстовый файл с расписанием находится в каталоге со всеми исходными файлами программы RConsole.



Администратор (пользователь устройством) напрямую через блокнот может менять содержимое данного текстового файла. На данный момент содержимое следующее:







В 21:01 загрузилась первая библиотека.



В 21:02 загрузилась вторая библиотека.

Ранее были приведены фрагменты кода обеспечивающие проверку на формат даты, а также саму дату, чтобы предотвратить загрузку библиотек расписание которых стоит на дату или же время, которое уже прошло:

