**ПСП Лабораторная работа №7 Теория**

**Оглавление**

**[Теория 2](#_Toc26383)**

[Вопрос 1: Что такое Mailslot? 2](#_Toc13330)

[Вопрос 2: Перечень функций интерфейса Mailslot API 2](#_Toc24113)

[Вопрос 3: В каком (каких) направлениях осуществляется пересылка сообщений с помощью Mailsolt? Как можно организовать двунаправленную пересылку? 2](#_Toc14861)

[Вопрос 4: Имена MailSoft? Можно ли несколько одинаковых имён? 3](#_Toc11965)

[Вопрос 5: (продолжая Вопрос 4). Размеры передаваемых сообщений? 3](#_Toc13070)

[Вопрос 6: Форматы имени почтового ящика? 4](#_Toc19817)

**[Для более удобного повторения ИТОГИ главы: 5](#_Toc23839)**

**[Практика 6](#_Toc21398)**

[Листинг 1. ServerMS 7](#_Toc31128)

[Листинг 2. СlientMS 9](#_Toc15569)

# Теория

## Вопрос 1: Что такое Mailslot?

Почтовым ящиком (Mailslot) называется объект ядра операционной системы, который обеспечивает передачу данных от процессов-клиентов к процессам-серверам, выполняющимся на компьютерах в одной локальной сети. Процесс, создающий почтовый ящик называется ***сервером почтового ящика***.

Процессы, которые связываются с почтовым ящиком, называются ***клиентами почтового*** ящика.

Если простыми словами для понимания: Представь обычный почтовый ящик, синенький на столбах. Ты туда кидаешь письмо, после чего почтальон забирает его и на почте обрабатывают. Где:

Почта - сервер, ты -клиент, почтовый ящик, тот самый виртуальный файл, куда информация сгружается, почтальон - сам канал доставки до сервера.

**!!!ВАЖНО!!!** отметить, что **Mailslot** не гарантирует 100% доставку. Мы могли это увидеть в нашей лабе, где он мог ошибочно читать буфер по несколько раз, слишкомы бвстро приходили сообщения и он мог из-за этого ломаться или слишком быстрая обработка, что по несколько раз читал одно и то же.

## **Вопрос 2: Перечень функций интерфейса Mailslot API**

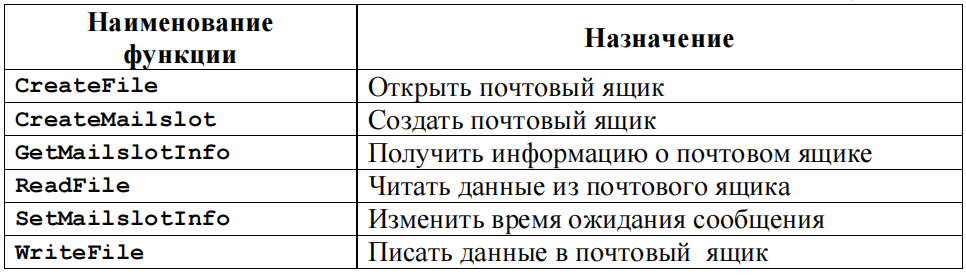


Рисунок 1. Функции интерфейса

## **Вопрос 3: В каком (каких) направлениях осуществляется пересылка сообщений с помощью Mailsolt? Как можно организовать двунаправленную пересылку?**

1. Передача может осуществляться только сообщениями и в одном направлении – от клиента к серверу. Обмен данными может происходить в синхронном и асинхронном режимах.
2. Нужно, чтобы сервер становился клиентом и искал существующие «Почтовые ящики», что создаст клиент, беря на себя роль сервера.

## **Вопрос 4: Имена MailSoft? Можно ли несколько одинаковых имён?**

1. Каждый почтовый ящик имеет имя, которое определяется сервером при создании и используется клиентами для доступа.
2. Допускается создание нескольких серверов с одинаковым именем почтового ящика – в этом случае все отправляемые клиентом сообщения будут поступать во все почтовые ящики, имеющие имя, указанное клиентом.

## **Вопрос 5: (продолжая Вопрос 4). Размеры передаваемых сообщений?**

Отправка сообщений возможна только в том случае, если их размер **не превышает 425 байт.**

Если клиент отправляет сообщение размером **менее 425 байт**, доставка не гарантируется.

Сообщения, **превышающие 425 байт**, могут быть отправлены только от одного клиента к одному серверу.

|  |
| --- |
| **Подробнее**  **Размер сообщений:**  MailSlot поддерживает ограничение на размер сообщений в 425 байт, чтобы обеспечить быструю и эффективную передачу данных. Это ограничение связано с архитектурой системы и необходимостью минимизации задержек при передаче.  Гарантия доставки:  Если сообщение меньше 425 байт, его доставка не гарантируется, потому что MailSlot не обеспечивает подтверждение получения для таких сообщений. Это значит, что сообщение может быть потеряно, если получающий процесс не активно слушает.  Отправка больших сообщений:  Сообщения, превышающие 425 байт, могут быть отправлены только от одного клиента к одному серверу, чтобы избежать путаницы и конфликтов при обработке данных. Это ограничение помогает обеспечить целостность и последовательность передачи данных.  **Применение:**  MailSlot часто используется для простых, однонаправленных сообщений, например, для уведомлений и сигналов между процессами. Эти ограничения делают MailSlot подходящим для сценариев, где требуется быстрый обмен небольшими сообщениями, но не критично важна доставка каждого сообщения. |

## **Вопрос 6: Форматы имени почтового ящика?**

**Локальный формат:** Используется при создании почтового ящика на локальном сервере. Клиент открывает ящик, если хочет работать с ящиками с одинаковым именем на одном компьютере.

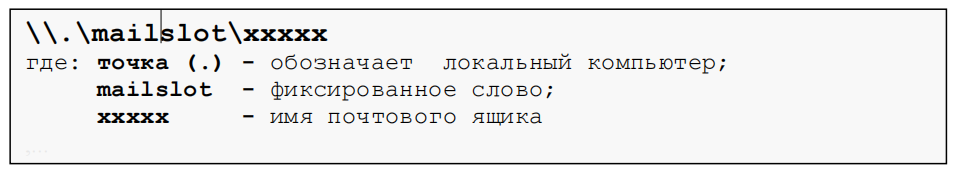


Рисунок 2. Локальный формат

**Сетевой формат:** Применяется для отправки сообщений в группу одноименных почтовых ящиков, расположенных на конкретном компьютере, указанном в имени.

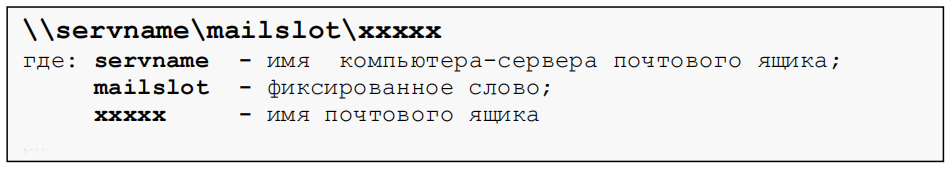


Рисунок 3. Сетевой формат

**Доменный формат:** Используется для отправки сообщений в группу одноименных почтовых ящиков на всех компьютерах в указанном домене. Если нужно отправить сообщения только на компьютеры в основном домене, можно использовать символ \* вместо имени домена.

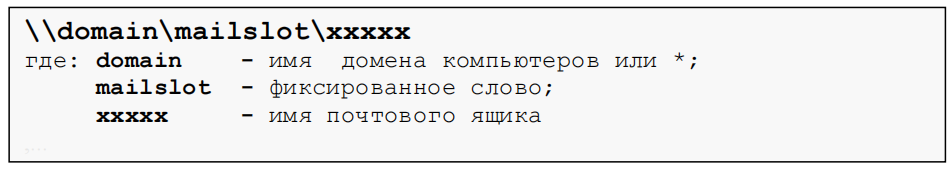


Рисунок 4. Доменный формат

**Разница:** **Сетевой формат** ориентирован на один конкретный компьютер, а **доменный** — на все компьютеры в определённом домене.

# Для более удобного повторения ИТОГИ главы:

1. Механизм Mailslots (почтовый ящик) является одним из IPC - механизмов операционной системы Windows, позволяющий создавать распределенные приложения архитектуры клиент-сервер в локальной сети TCP/IP.
2. Почтовый ящик представляет собой объект операционной системы, предоставляющий возможность пересылать данные в одном направлении: от клиента к серверу.
3. Почтовый ящик идентифицируется своим именем. Сервером называется процесс создающий почтовый ящик. Клиентом – процесс, который подключается к почтовому ящику и записывает в него данные.
4. Обмен данными осуществляется сообщениями и может происходить в синхронном и асинхронном режимах. Если клиент и сервер находятся на разных компьютерах, доставка сообщений не гарантируется.
5. Допускается создание нескольких ящиков с одним и тем же именем. Если пересылаемые сообщения не превышают 425 байт, то возможна передача данных одновременно нескольким почтовым ящикам.
6. В состав Mailslots API входят функции создания почтового ящика, подсоединения клиента к почтовому ящику, функции записи и чтения сообщений, а также функции для получения и установки характеристик почтового ящика.

# Практика

|  |
| --- |
| #include <windows.h> // Заголовочный файл для работы с Windows API, включая функции для Mailslot.  #include <iostream> // Заголовочный файл для работы с вводом/выводом в консоли.  #include <ctime> // Заголовочный файл для работы с функциями времени.  using namespace std; // Пространство имён std для использования стандартных функций и объектов.  // Функция для обработки ошибок и вывода сообщений об ошибке.  void SetErrorMail(const char\* msg, DWORD errorCode) {  cerr << msg << ": " << errorCode << endl; // Вывод сообщения об ошибке в стандартный поток ошибок.  exit(errorCode); // Завершение программы с указанным кодом ошибки.  }  int main() {  setlocale(LC\_ALL, "rus"); // Устанавливаем локаль для корректного отображения русских символов.  // Создание Mailslot для получения сообщений от клиентов.  HANDLE serverMailSlot = CreateMailslot(  L"\\\\.\\mailslot\\Box1", // Имя Mailslot. Оно включает полный путь и уникальное имя (\\.\\mailslot\\Box1).  500, // Максимальный размер сообщения в байтах (500 байт).  180000, // Таймаут ожидания в миллисекундах (180000 = 3 минуты).  NULL // Указатель на SECURITY\_ATTRIBUTES (NULL = настройки по умолчанию).  );  // Проверяем, удалось ли создать Mailslot.  if (serverMailSlot == INVALID\_HANDLE\_VALUE) {  SetErrorMail("Ошибка создания Mailslot", GetLastError()); // Если ошибка, выводим сообщение и завершаем программу.  }  cout << "Сервер запущен и ожидает сообщения..." << endl; // Уведомляем пользователя о запуске сервера.  char readBuf[500]; // Буфер для хранения получаемых данных (длина соответствует максимальному размеру сообщения).  DWORD bytesRead; // Переменная для хранения количества прочитанных байт.  int messageCount = 0; // Счётчик сообщений, ограничиваем до 1000 сообщений.  while (messageCount < 1000) { // Цикл для приёма сообщений до достижения 1000.  if (ReadFile(serverMailSlot, readBuf, sizeof(readBuf), &bytesRead, NULL)) { // Читаем данные из Mailslot.  /\*  Параметры ReadFile:  1. serverMailSlot - дескриптор файла (в данном случае Mailslot).  2. readBuf - указатель на буфер, куда записываются данные.  3. sizeof(readBuf) - размер буфера, доступного для записи.  4. &bytesRead - указатель на переменную для записи количества прочитанных байт.  5. NULL - указатель на OVERLAPPED структуру (NULL = синхронный режим).  \*/  cout << "Сообщение от клиента №" << messageCount << " - " << readBuf << endl; // Выводим сообщение в консоль.  messageCount++; // Увеличиваем счётчик сообщений.  }  else {  cerr << "Ошибка чтения сообщения: " << GetLastError() << endl; // Если чтение не удалось, выводим ошибку.  }  }  CloseHandle(serverMailSlot); // Закрываем дескриптор Mailslot после завершения работы.  return 0; // Возвращаем 0 как код успешного завершения программы.  } |

## Листинг 1. ServerMS

В листинге 1. Код с искусственным ограничением количества принимаемых сообщений. Это сделано для попытки избежать ошибки. В кодах ServerMS2 и ServerMS3 был убран цикл. В ServerMS2 была попытка менять буфер и время, ServerMS3 соответствует изменениям, требуемым в последних заданиях лабораторной.

|  |
| --- |
| #include <iostream> // - iostream для работы с вводом/выводом;  #include <windows.h> // - windows.h для работы с функциями API Windows, такими как CreateFile и WriteFile;  #include <string> // - string для работы со строками;  #include <ctime> // - ctime для измерения времени выполнения.  using namespace std;  // Функция для получения текстового описания ошибки по коду.  string GetErrorMail(int code) {  string msgText = "";  switch (code) {  case WSAEINTR: msgText = "WSAEINTR"; break; // Операция была прервана.  case WSAEACCES: msgText = "WSAEACCES"; break; // Доступ запрещён.  case WSAEFAULT: msgText = "WSAEFAULT"; break; // Ошибка указателя.  default: msgText = "Error"; break; // Общее сообщение об ошибке.  }  return msgText;  }  // Функция объединяет текст сообщения и описание ошибки.  string SetErrorMail(string msgText, int code) {  return msgText + GetErrorMail(code); // Возвращает полное сообщение об ошибке.  }  int main() {  setlocale(LC\_ALL, "rus"); // Установка русской локали для корректного отображения текста на русском.  try {  HANDLE clientMailSlot1, clientMailSlot2, clientMailSlot3; // Переменные для описателей (дескрипторов) Mailslot.  double t1, t2; // Переменные для измерения времени выполнения передачи сообщений.  // Подключение к Mailslot BOX1 на удалённом сервере.  if ((clientMailSlot1 = CreateFile(  L"\\\\DESKTOP-DLOVPOH\\mailslot\\BOX1", // Путь к Mailslot. L перед строкой указывает на использование Unicode.  GENERIC\_WRITE, // Режим доступа: запись.  FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, // Совместный доступ: чтение и запись.  NULL, // Указатель на структуру безопасности (NULL — использовать настройки по умолчанию).  OPEN\_EXISTING, // Указывает, что Mailslot уже должен существовать.  NULL, // Флаги и атрибуты файла (NULL — параметры по умолчанию).  NULL // Шаблон файла (NULL — не используется).  )) == INVALID\_HANDLE\_VALUE) {  throw SetErrorMail("CreateFile (BOX1): ", GetLastError()); // Генерация исключения, если дескриптор недействителен.  }  // Подключение к Mailslot BOX2.  if ((clientMailSlot2 = CreateFile(  L"\\\\DESKTOP-DLOVPOH\\mailslot\\BOX2", GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, NULL, NULL)) == INVALID\_HANDLE\_VALUE) {  throw SetErrorMail("CreateFile (BOX2): ", GetLastError());  }  // Подключение к Mailslot BOX3.  if ((clientMailSlot3 = CreateFile(  L"\\\\DESKTOP-DLOVPOH\\mailslot\\BOX3", GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, NULL, NULL)) == INVALID\_HANDLE\_VALUE) {  throw SetErrorMail("CreateFile (BOX2): ", GetLastError());  }  cout << "Hello, I am Client" << endl; // Сообщение о начале работы клиента.  char writeBuf[50] = "Hello from Client-Mailslot"; // Буфер для записи сообщения.  DWORD writeMsg; // Переменная для хранения количества записанных байтов.  t1 = clock(); // Засекаем начальное время выполнения.  // Цикл для отправки 1000 сообщений на серверы.  for (int i = 1; i <= 1000; i++) {  if (!WriteFile(clientMailSlot1, writeBuf, sizeof(writeBuf), &writeMsg, NULL)) {  throw SetErrorMail("WriteFile (BOX1): ", GetLastError()); // Исключение при ошибке записи в BOX1.  }  if (!WriteFile(clientMailSlot2, writeBuf, sizeof(writeBuf), &writeMsg, NULL)) {  throw SetErrorMail("WriteFile (BOX2): ", GetLastError()); // Исключение при ошибке записи в BOX2.  }  if (!WriteFile(clientMailSlot3, writeBuf, sizeof(writeBuf), &writeMsg, NULL)) {  throw SetErrorMail("WriteFile (BOX2): ", GetLastError()); // Исключение при ошибке записи в BOX3.  }  cout << "Message " << i << " was sent to both servers" << endl; // Вывод информации об отправленном сообщении.  }  t2 = clock(); // Засекаем конечное время выполнения.  // Закрытие дескрипторов.  if (!CloseHandle(clientMailSlot1)) {  throw "Error: CloseHandle (BOX1)"; // Исключение при ошибке закрытия дескриптора BOX1.  }  if (!CloseHandle(clientMailSlot2)) {  throw "Error: CloseHandle (BOX2)"; // Исключение при ошибке закрытия дескриптора BOX2.  }  if (!CloseHandle(clientMailSlot3)) {  throw "Error: CloseHandle (BOX2)"; // Исключение при ошибке закрытия дескриптора BOX3.  }  // Вывод времени передачи сообщений.  cout << endl << "Время передачи: " << (t2 - t1) / 1000 << " сек." << endl << endl;  system("pause"); // Ожидание нажатия клавиши для завершения программы.  }  catch (string e) { // Обработка исключений.  cout << e << endl; // Вывод сообщения об ошибке.  }  return 0; // Возвращение успешного завершения программы.  } |

## Листинг 2. СlientMS