Лаба6

Теория

Современные операционные системы имеют встроенные механизмы межпроцессного взаимодействия - IPC (InterProcess Communication), предназначенные для обмена данными между процессами и для синхронизации процессов.

Разработчики программного обеспечения могут использовать IPC с помощью предоставляемых операционными системами программных интерфейсов (API).

Программный интерфейс одного из IPC-механизмов операционной системы Windows, который может быть использован для обмена данными между распределенными в локальной сети процессами, и имеет название Named Pipe (именованный канал).

Именованным каналом называется объект ядра операционной системы, который обеспечивает обмен данными между процессами, выполняющимися на компьютерах в одной локальной сети. Процесс, создающий именованный канал, называется сервером именованного канала. Процессы, которые связываются с именованным каналом, называются клиентами именованного канала. Любой именованный канал идентифицируется своим именем, которое задается при создании канала.

Именованные каналы бывают: дуплексные (позволяющие передавать данные в обе стороны) и полудуплексные (позволяющие передавать данные только в одну сторону). Передача данных в именованном канале может осуществляться как потоком, так и сообщениями. Обмен данными в канале может быть синхронным и асинхронным.

Сами функции интерфейса располагаются в библиотеке KERNEL32.DLL ядра операционной системы.

Все функции Named Pipe API можно разбить на три группы: функции управления каналом (создать канал, соединить сервер с каналом, открыть канал, получить информацию об именованном канале, получить состояние канала, изменить характеристики канала); функции обмена данными (писать в канал, читать из канала, копировать данные канала) и функции для работы с транзакциями.

```
\\.\pipe\xxxxx
где: точка (.) - обозначает  локальный компьютер;
     pipe      - фиксированное слово;
     xxxxx     - имя канала
```

Рисунок 4.3.2. Локальный формат имени канала

```
\\servname\pipe\xxxxx
где:  servname - имя компьютера – сервера именованного канала;
      pipe     - фиксированное слово;
      xxxxx    - имя канала
```

Рисунок 4.3.3. Сетевой формат имени канала

При использовании форматов имени канала, необходимо помнить, что:
1) при создании канала всегда используется локальный формат имени;
2) если клиент удаленный (на другом компьютере), то он всегда должен использовать сетевой формат имени; при этом обмен данными между клиентом и сервером осуществляется сообщениями;
3) если клиент локальный и использует сетевой формат имени при подсоединении к каналу (функция CreateFile), то обмен данными осуществляется сообщениями;
4) если клиент локальный и использует локальный формат имени канала, то обмен данными осуществляется потоком.

Итоги

1. Современные операционные системы имеют встроенные механизмы межпроцессорного взаимодействия (IPC), позволяющие создавать распределенные в локальной сети приложения. Для работы использования IPC-механизмов операционные системы предоставляют специальные программные интерфейсы.
2. Интерфейс Named Pipe (именованный канал) реализует один из IPC-механизмов операционной системы Windows и позволяет создавать распределенные приложения архитектуры клиент-сервер.
3. Именованный канал представляет собой объект операционной системы Windows, позволяющий создавать между распределенными в локальной TCP/IP-сети процессами дуплексные и полудуплексные каналы, по которым может осуществляться передача данных в синхронном или асинхронном режимах.
4. В состав интерфейса Named Pipe входят функции для управления каналом, функции для обмена данными по каналу и функции работы с транзакциями.

Задание 1 и 2:

Создание серверного и клиентского приложения ServerNP и ClientNP:

1. Цель задания:

Реализовать серверное и клиентское приложения, которые используют именованный канал для обмена данными. В сервере создается канал CreateNamedPipe, клиент подключается к нему через CreateFile.
Реализованы функции ConnectNamedPipe, ReadFile, WriteFile в сервере для соединения с клиентом и обмена данными.
2. Работа кода:
ServerNP:
Создает именованный канал с именем \\\\.\\pipe\\Tube с использованием CreateNamedPipe.
Ожидает подключения клиента с помощью ConnectNamedPipe.
После подключения обрабатывает данные, поступающие от клиента, с использованием ReadFile. Если сообщение равно STOP, сервер прекращает работу. Если сообщение получено, сервер выводит его и отправляет обратно клиенту с помощью WriteFile.
Завершает работу с каналом с использованием DisconnectNamedPipe и CloseHandle.
ClientNP:
Подключается к каналу с тем же именем, что и на сервере, с помощью CreateFile.
Читает и пишет сообщения в канал с использованием ReadFile и WriteFile.
Клиент и сервер взаимодействуют в синхронном режиме.
Запуск:
Сначала запускается ServerNP. Он переходит в состояние ожидания соединения.
Затем запускается ClientNP, который подключается к серверу и инициирует обмен данными.
Приложения можно проверить, отправив разные сообщения и убедившись, что сервер корректно их обрабатывает.
Задание 3:
Обмен данными между сервером и клиентом
Выполнение:
Задание выполнено за счет использования ReadFile и WriteFile в ServerNP и ClientNP.
Сервер ожидает входящие сообщения от клиента и отвечает обратно. Клиент также обрабатывает сообщения от сервера.
Задание 4:
Реализация блоков 2 и 3
ServerNP и ClientNP:
Реализованы блоки обработки данных: сервер принимает сообщения, проверяет их содержимое и отправляет обратно. Клиент отправляет сообщения серверу и ожидает ответ.
Используются ReadFile и WriteFile для взаимодействия.
Задание 5 и 6:
Взаимодействие через локальную сеть
ClientNP:

Для взаимодействия по сети предусмотрено использование PIPE_NAME_LAN. Канал с этим именем позволяет клиенту подключаться к серверу на другом компьютере в локальной сети.
Использование сетевого имени реализовано, что позволяет проверять подключение через локальную сеть.

Задание 7:

ClientNPt с использованием TransactNamedPipe

1. Цель задания:

Реализовать клиент, который взаимодействует с сервером с использованием TransactNamedPipe вместо ReadFile и WriteFile.

2. Работа кода:

ClientNPt подключается к каналу и отправляет сообщение с использованием TransactNamedPipe. Это позволяет выполнять атомарный обмен данных (запрос-ответ) за один вызов.
При успешном выполнении TransactNamedPipe клиент отправляет сообщение и получает ответ от сервера.

Запуск:

Сначала запускается сервер ServerNP.
Затем запускается ClientNPt, который подключается к серверу и выполняет обмен сообщениями.

Задание 8:

ClientNPct с использованием CallNamedPipe

1. Цель задания:

Реализовать клиент, который взаимодействует с сервером с использованием CallNamedPipe.

2. Работа кода:

ClientNPct вызывает CallNamedPipe для отправки сообщений серверу и получения ответа. Это позволяет упростить взаимодействие, объединяя создание, чтение и запись в одном вызове.
Использование CallNamedPipe также позволяет более эффективно взаимодействовать с именованным каналом, не требуя явного управления соединениями.

ОСНОВНЫЕ ФУНКЦИИ С ПАРАМЕТРАМИ

1. CreateNamedPipe

Назначение: Создает и инициализирует именованный канал. Сервер использует эту функцию для определения нового канала, через который будет происходить обмен данными.

Синтаксис:

```
HANDLE CreateNamedPipe(
LPCWSTR lpName, // Имя канала
DWORD dwOpenMode,  // Режим доступа (чтение/запись)
DWORD dwPipeMode,  // Режим канала (сообщение/байтовый,
синхронный/асинхронный)
DWORD nMaxInstances,  // Максимальное количество экземпляров
```

DWORD nOutBufferSize, // Размер выходного буфера
DWORD nInBufferSize, // Размер входного буфера
DWORD nDefaultTimeOut,// Время ожидания соединения
LPSECURITY_ATTRIBUTES lpSecurityAttributes //Атрибуты безопасности
);

Параметры:

lpName: Имя канала (например, \\\\.\\pipe\\Tube).
dwOpenMode: Указывает режим доступа (например, PIPE_ACCESS_DUPLEX для дуплексного обмена).
dwPipeMode: Определяет способ передачи сообщений и синхронность (например, PIPE_TYPE_MESSAGE | PIPE_WAIT).
nMaxInstances: Максимальное количество одновременно существующих экземпляров канала.

Описание: Создает канал и возвращает дескриптор, который будет использоваться для операций чтения/записи и подключения.

2. ConnectNamedPipe

Назначение: Ожидает подключения клиента к серверу через именованный канал.

Синтаксис:

BOOL ConnectNamedPipe(
HANDLE hNamedPipe,// Дескриптор канала
LPOVERLAPPED lpOverlapped // Перекрывающая структура (не используется в синхронном режиме)
);

Параметры:

hNamedPipe: Дескриптор, возвращенный функцией CreateNamedPipe.
lpOverlapped: Указатель на структуру OVERLAPPED для асинхронных операций (необходим для асинхронных каналов).

Описание: Блокирует выполнение до тех пор, пока клиент не подключится к каналу.

3. CreateFile

Назначение: Открывает соединение с существующим именованным каналом, используется клиентом для подключения к серверу.

Синтаксис:

HANDLE CreateFile(
LPCWSTR lpFileName, // Имя или путь к файлу (имя канала)
DWORD dwDesiredAccess, // Режим доступа (чтение/запись)
DWORD dwShareMode,// Режим совместного доступа
LPSECURITY_ATTRIBUTES lpSecurityAttributes, // Атрибуты безопасности
DWORD dwCreationDisposition, // Действие при создании файла
DWORD dwFlagsAndAttributes,// Атрибуты и флаги файла

HANDLE hTemplateFile // Шаблон для создания файла (не используется для именованных каналов)

);

Параметры:

lpFileName: Имя канала (например, \\\\.\\pipe\\Tube).

dwDesiredAccess: Режим доступа (например, GENERIC_READ | GENERIC_WRITE).

Описание: Открывает дескриптор для работы с каналом.

4. ReadFile

Назначение: Считывает данные из именованного канала.

Синтаксис:

BOOL ReadFile(

HANDLE hFile, // Дескриптор канала

LPVOID lpBuffer, // Буфер для хранения данных

DWORD nNumberOfBytesToRead, // Количество байт для чтения

LPDWORD lpNumberOfBytesRead, // Фактически прочитанные байты

LPOVERLAPPED lpOverlapped // Перекрывающая структура (для асинхронного режима)

);

Параметры:

hFile: Дескриптор канала, с которым связано чтение.

lpBuffer: Указатель на буфер для данных.

Описание: Блокирует выполнение до тех пор, пока данные не будут считаны или не произойдет ошибка.

5. WriteFile

Назначение: Записывает данные в именованный канал.

Синтаксис:

BOOL WriteFile(

HANDLE hFile, // Дескриптор канала

LPCVOID lpBuffer,// Буфер с данными для записи

DWORD nNumberOfBytesToWrite, // Количество байт для записи

LPDWORD lpNumberOfBytesWritten, // Фактически записанные байты

LPOVERLAPPED lpOverlapped // Перекрывающая структура (для асинхронного режима)

);

Параметры:

hFile: Дескриптор канала.

lpBuffer: Указатель на буфер с данными для записи.

Описание: Блокирует выполнение до тех пор, пока данные не будут записаны.

6. TransactNamedPipe

Назначение: Выполняет атомарную операцию чтения и записи в именованном канале. Клиент может одновременно отправить запрос и получить ответ.

Синтаксис:

```
BOOL TransactNamedPipe(
HANDLE hNamedPipe,  // Дескриптор канала
LPVOID lpInBuffer,  // Входные данные
DWORD nInBufferSize,// Размер входных данных
LPVOID lpOutBuffer, // Выходные данные
DWORD nOutBufferSize,  // Размер выходных данных
LPDWORD lpBytesRead,// Количество прочитанных байт
LPOVERLAPPED lpOverlapped // Перекрывающая структура (для
асинхронного режима)
);
```

Описание: Полезна для упрощения обмена данными без отдельного вызова ReadFile и WriteFile.

7. CallNamedPipe

Назначение: Выполняет обмен данными между клиентом и сервером через именованный канал, аналогично TransactNamedPipe, но без необходимости явного открытия и закрытия канала клиентом.

Синтаксис:

```
BOOL CallNamedPipe(
LPCWSTR lpNamedPipeName,  // Имя канала
LPVOID lpInBuffer,  // Входные данные
DWORD nInBufferSize,// Размер входных данных
LPVOID lpOutBuffer, // Выходные данные
DWORD nOutBufferSize,  // Размер выходных данных
LPDWORD lpBytesRead,// Количество прочитанных байт
DWORD nTimeOut// Время ожидания
);
```

Описание: Упрощает взаимодействие клиента с сервером через именованный канал за счет объединения создания, отправки и закрытия в одном вызове.

8. SetNamedPipeHandleState

Назначение: Устанавливает режим работы для уже существующего дескриптора именованного канала. Эта функция полезна для изменения способа обмена данными между клиентом и сервером.

Синтаксис:

```
BOOL SetNamedPipeHandleState(
HANDLE hNamedPipe,  // Дескриптор канала
LPDWORD lpMode,  // Указатель на новый режим чтения/записи
LPDWORD lpMaxCollectionCount,// Максимальное количество
сообщений для коллекции (необязательный параметр)
```

LPDWORD lpCollectDataTimeout // Таймаут для сбора данных (необязательный параметр)
);

Параметры:

hNamedPipe: Дескриптор именованного канала.

lpMode: Указатель на новый режим работы, например, PIPE_READMODE_MESSAGE для установки режима сообщений или PIPE_READMODE_BYTE для побайтового режима.

lpMaxCollectionCount и lpCollectDataTimeout: Опциональные параметры для управления коллекцией сообщений и сбором данных.

Описание: Эта функция особенно полезна для клиентов, которые могут изменять способ чтения данных, например, переключаться между режимами сообщений и побайтового обмена.

9. DisconnectNamedPipe

Назначение: Разрывает существующее подключение с клиентом. Сервер использует эту функцию для завершения связи с клиентом после обработки его запросов.

Синтаксис:

BOOL DisconnectNamedPipe(
HANDLE hNamedPipe // Дескриптор именованного канала
);

Параметры:

hNamedPipe: Дескриптор именованного канала, который нужно отключить.

Описание: Отключает текущего клиента от канала, чтобы сервер мог ожидать новое подключение. Эта функция полезна для серверов, обрабатывающих несколько клиентов.

10. PeekNamedPipe

Назначение: Позволяет просматривать данные, доступные в канале, без удаления их из буфера. Это полезно для проверки содержимого данных перед их фактическим чтением.

Синтаксис:

BOOL PeekNamedPipe(
HANDLE hNamedPipe, // Дескриптор канала
LPVOID lpBuffer, // Буфер для данных
DWORD nBufferSize, // Размер буфера
LPDWORD lpBytesRead,// Количество прочитанных байт
LPDWORD lpTotalBytesAvail,// Доступные байты в канале
LPDWORD lpBytesLeftThisMessage // Оставшиеся байты текущего сообщения
);

Параметры:

hNamedPipe: Дескриптор канала, из которого производится просмотр данных.
lpBuffer: Указатель на буфер для чтения данных.
nBufferSize: Размер буфера.
lpBytesRead: Указатель на количество фактически прочитанных байт.
lpTotalBytesAvail: Указатель на количество доступных байт в канале.
lpBytesLeftThisMessage: Указатель на количество оставшихся байт текущего сообщения.

Описание: Функция позволяет заглянуть в данные канала без их удаления, что полезно для предварительной обработки данных.

11. WaitNamedPipe

Назначение: Ожидает, пока именованный канал будет готов к подключению. Используется клиентами для проверки доступности канала перед подключением.

Синтаксис:

```
BOOL WaitNamedPipe(
LPCWSTR lpNamedPipeName,  // Имя канала
DWORD nTimeOut// Таймаут ожидания (в миллисекундах)
);
```

Параметры:

lpNamedPipeName: Имя канала, к которому пытается подключиться клиент.
nTimeOut: Время ожидания доступности канала (в миллисекундах).

Описание: Эта функция позволяет клиенту ждать, пока канал станет доступным, в случае если он занят или еще не создан сервером.

12. GetNamedPipeInfo

Назначение: Возвращает информацию об указанном именованном канале.

Синтаксис:

```
BOOL GetNamedPipeInfo(
HANDLE hNamedPipe,  // Дескриптор канала
LPDWORD lpFlags, // Указатель на флаги (режимы работы)
LPDWORD lpOutBufferSize,  // Размер выходного буфера
LPDWORD lpInBufferSize,// Размер входного буфера
LPDWORD lpMaxInstances // Максимальное количество экземпляров
);
```

Параметры:

hNamedPipe: Дескриптор именованного канала.
lpFlags: Указатель на флаги, описывающие текущие настройки канала.
lpOutBufferSize и lpInBufferSize: Указатели на размеры выходного и входного буфера.
lpMaxInstances: Указатель на максимальное количество экземпляров канала.

Описание: Предоставляет информацию о канале, такую как его режим, размер буфера и максимальное количество экземпляров. Полезно для отладки и мониторинга работы канала.

Лаба7

Теория

IPC — механизм, поддерживаемый операционной системой Windows и имеющий название Mailslots (почтовый ящик). Также как и Named Pipe механизм Mailslots может быть использован для обмена данными между распределенными в локальной сети процессами.

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.

Каждый почтовый ящик имеет имя, которое определяется сервером при создании и используется клиентами для доступа. Передача может осуществляться только сообщениями и в одном направлении — от клиента к серверу. Обмен данными может происходить в синхронном и асинхронном режимах. 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. Однако, следует сказать, что такая рассылка сообщений возможна только в том случае, когда длина отправляемых сообщений не превышает 425 байт.

```
\\.\mailslot\xxxxx
где: точка (.) - обозначает  локальный компьютер;
     mailslot  - фиксированное слово;
     xxxxx     - имя почтового ящика
```

Рисунок 5.3.2. Локальный формат имени почтового ящика

Локальный формат имени почтового ящика используется при создании почтового ящика (ящик всегда создается на локальном для сервера компьютере), а также программой клиентом при открытии ящика, если предполагается использовать для записи все ящики с заданным именем на одном локальном компьютере.

Сетевой формат имени почтового ящика используется программой клиента, для записи сообщений в группу одноименных почтовых ящиков, которые находятся на компьютере, указанном в имени.

```
\\servname\mailslot\xxxxx
где: servname  - имя  компьютера-сервера почтового ящика;
     mailslot  - фиксированное слово;
     xxxxx     - имя почтового ящика
```

Рисунок 5.3.3. Сетевой формат имени почтового ящика

```
\\domain\mailslot\xxxxx
где: domain    - имя  домена компьютеров или *;
     mailslot  - фиксированное слово;
     xxxxx     - имя почтового ящика
```

Рисунок 5.3.4. Доменный формат имени почтового ящика

Доменный формат имени почтового ящика используется программой клиента для записи сообщений в группу одноименных почтовых ящиков, которые находятся на всех компьютерах указанного домена. Если необходимо записать в сообщение в группу почтовых ящиков, которые находятся на компьютерах первичного домена, то вместо имени домена можно указать символ *.

Итоги

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.

1. Введение и назначение кода

Код реализует простую систему обмена сообщениями с использованием Mailslots в Windows. Основная цель состоит в передаче сообщений от клиента к серверу через механизм Mailslots. ClientMS отправляет сообщение в Mailslot, а ServerMS принимает его. Mailslots позволяют асинхронное, однонаправленное взаимодействие, обычно используются для простого обмена сообщениями в локальной сети.

2. Запуск кода

ServerMS следует запустить первым, так как он создает Mailslot для получения сообщений.

После запуска ServerMS можно запустить ClientMS, который отправляет сообщения на Mailslot, созданный сервером.

3. Принцип работы и разбор кода

ServerMS

Создание Mailslot: Сервер создает Mailslot с именем \\.\mailslot\Box с помощью функции CreateMailslot.

```
hM = CreateMailslot(L"\\\\.\\mailslot\\Box", 300,
MAILSLOT_WAIT_FOREVER, NULL);
```

L"\\\\.\\mailslot\\Box" путь к Mailslot. Все Mailslots в Windows используют формат \\.\mailslot\имя.

300 максимальный размер сообщения.

MAILSLOT_WAIT_FOREVER ожидание на чтение сообщения (блокирует поток до поступления данных).

Чтение сообщений: Сервер циклически вызывает ReadFile для чтения сообщений из Mailslot.

```
for (int i = 0; i <= 1000; i++) {
    if (!ReadFile(hM, rbuf, sizeof(rbuf), &rb, NULL))
    throw "ReadFileError";
cout << "Сообщение от клиента: " << rbuf << " " << i << endl;
}
```

Цикл на 1000 итераций ожидает сообщения.

rbuf буфер для приема сообщений.

Если ReadFile возвращает ошибку, выбрасывается исключение.

Измерение времени: Время работы кода измеряется с помощью clock(), которое замеряет интервал выполнения программы.

Закрытие дескриптора: CloseHandle закрывает дескриптор Mailslot после завершения работы.

ClientMS

Подключение к Mailslot: Клиент открывает существующий Mailslot с помощью CreateFile для записи сообщений.

```
hM = CreateFile(L"\\\\.\\mailslot\\Box", GENERIC_WRITE,
FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL);
```

Путь L"\\\\.\\mailslot\\Box" соответствует имени Mailslot, созданному сервером.

GENERIC_WRITE указывает на возможность записи.

Отправка сообщений: Клиент циклически отправляет сообщения через WriteFile.

```
for (int i = 0; i <= 1000; i++) {
    if (!WriteFile(hM, wbuf, sizeof(wbuf), &wb, NULL))
    throw "ReadFileError";
cout << wbuf << " " << i << endl;
}
```

wbuf содержит сообщение "Hello Mailslot".

Каждое сообщение передается серверу.

Закрытие дескриптора: CloseHandle завершает работу с Mailslot.

4. Важные детали и принципы работы

Асинхронное взаимодействие: Mailslots не обеспечивают гарантированной доставки сообщений, но удобны для передачи данных в локальной сети. Сервер ожидает сообщения, пока клиент их отправляет.

Блокирующие вызовы: Вызов ReadFile блокирует выполнение, пока данные не будут переданы клиентом.

Однонаправленность: В данной реализации данные передаются в одном направлении: от клиента к серверу. Для обратной передачи сообщений нужно создать второй Mailslot.

Обработка ошибок: Программы используют try-catch блоки для обработки ошибок, связанных с функциями CreateMailslot, CreateFile, ReadFile, WriteFile.

5. Алгоритм выполнения

Сервер (ServerMS)

Создает Mailslot.

Ожидает поступления сообщений от клиента.

Читает и выводит сообщения на экран.

Завершает работу.

Клиент (ClientMS)

Открывает соединение с созданным сервером Mailslot.

Отправляет 1000 сообщений с текстом "Hello Mailslot".

Завершает работу.

ОСНОВНЫЕ ФУНКЦИИ С ПАРАМЕТРАМИ

1. CreateMailslot

Предназначение:

Создает Mailslot это односторонний канал связи, используемый для получения сообщений. Он доступен для всех процессов в локальной сети.

Сигнатура:

```
HANDLE CreateMailslot(
LPCWSTR lpName,          // Имя Mailslot
DWORD nMaxMessageSize,   // Максимальный размер сообщения (в байтах)
DWORD lReadTimeout,      // Тайм-аут ожидания чтения (в миллисекундах)
```

LPSECURITY_ATTRIBUTES lpSecurityAttributes // Атрибуты безопасности
);

Параметры:

lpName: Указывает имя Mailslot. Например, L"\\\\.\\mailslot\\Box". Формат \\.\mailslot\<имя> используется для локального создания Mailslot.

nMaxMessageSize: Устанавливает максимальный размер сообщения в байтах. Указание 0 означает, что ограничение на размер отсутствует.

lReadTimeout: Тайм-аут ожидания в миллисекундах. Значение MAILSLOT_WAIT_FOREVER заставляет сервер ожидать, пока сообщение не поступит.

lpSecurityAttributes: Указывает атрибуты безопасности. Если передается NULL, используется стандартная конфигурация безопасности.

Возвращаемое значение:

Возвращает дескриптор Mailslot. Если функция завершается с ошибкой, возвращает INVALID_HANDLE_VALUE.

Что выполняет:

Создает Mailslot с указанными параметрами.

Этот дескриптор используется для чтения сообщений, отправляемых клиентами.

2. CreateFile

Предназначение:

Открывает существующий объект, например файл, устройство или, как в данном случае, Mailslot для записи или чтения.

Сигнатура:

HANDLE CreateFile(
LPCWSTR lpFileName, // Имя или путь к объекту (например, Mailslot)
DWORD dwDesiredAccess, // Уровень доступа (чтение/запись)
DWORD dwShareMode, // Разделяемый доступ
LPSECURITY_ATTRIBUTES lpSecurityAttributes, // Атрибуты безопасности
DWORD dwCreationDisposition, // Поведение создания/открытия
DWORD dwFlagsAndAttributes, // Флаги и атрибуты файла
HANDLE hTemplateFile // Шаблон файла (не используется)
);

Параметры:

lpFileName: Указывает имя Mailslot, с которым клиент будет взаимодействовать (например, L"\\\\.\\mailslot\\Box").

dwDesiredAccess: Указывает права доступа. В нашем случае GENERIC_WRITE для клиента.

dwShareMode: Указывает, как файл или объект может совместно использоваться другими процессами. Обычно FILE_SHARE_READ для Mailslot.

lpSecurityAttributes: Если указано NULL, используются стандартные атрибуты безопасности.
dwCreationDisposition: Задает, что делать с существующими файлами/объектами. В случае с Mailslot используется OPEN_EXISTING.
dwFlagsAndAttributes: Не используется для Mailslot (можно передать 0).
hTemplateFile: Не используется для Mailslot (можно передать NULL).

Возвращаемое значение:

Возвращает дескриптор, указывающий на открытый объект, в данном случае Mailslot. Если произошла ошибка, возвращается INVALID_HANDLE_VALUE.

Что выполняет:

Открывает Mailslot для записи сообщений клиентом.

3. ReadFile

Предназначение:

Читает данные из файла или устройства, определенного дескриптором, в этом случае из Mailslot, созданного сервером.

Сигнатура:

```
BOOL ReadFile(
HANDLE hFile,              // Дескриптор файла или Mailslot
LPVOID lpBuffer,           // Буфер для хранения прочитанных данных
DWORD nNumberOfBytesToRead, // Размер данных для чтения
LPDWORD lpNumberOfBytesRead, // Указатель на переменную, хранящую количество прочитанных байт
LPOVERLAPPED lpOverlapped  // Используется для асинхронного ввода-вывода (NULL для синхронного)
);
```

Параметры:

hFile: Дескриптор, возвращенный функцией CreateMailslot.
lpBuffer: Указатель на буфер, в который будут записаны данные.
nNumberOfBytesToRead: Количество байт, которые нужно прочитать.
lpNumberOfBytesRead: Указатель на переменную, которая будет содержать фактически прочитанное количество байт.
lpOverlapped: Используется для асинхронного ввода-вывода. В данном случае NULL, так как чтение синхронное.

Возвращаемое значение:

Возвращает TRUE при успешном чтении данных и FALSE при ошибке.

Что выполняет:

Читает данные, отправленные клиентом, из Mailslot.

Блокирует выполнение, пока данные не поступят, если Mailslot настроен на ожидание.

4. WriteFile

Предназначение:

Записывает данные в файл или устройство, определенное дескриптором, в данном случае отправляет сообщения в Mailslot.

Сигнатура:

```
BOOL WriteFile(
HANDLE hFile,            // Дескриптор файла или Mailslot
LPCVOID lpBuffer,        // Буфер с данными для записи
DWORD nNumberOfBytesToWrite, // Размер данных для записи
LPDWORD lpNumberOfBytesWritten, // Указатель на переменную, хранящую количество записанных байт
LPOVERLAPPED lpOverlapped  // Используется для асинхронного ввода-вывода (NULL для синхронного)
);
```

Параметры:

hFile: Дескриптор, возвращенный функцией CreateFile.
lpBuffer: Указатель на буфер с данными для записи.
nNumberOfBytesToWrite: Количество байт для записи.
lpNumberOfBytesWritten: Указатель на переменную, которая будет содержать количество фактически записанных байт.
lpOverlapped: Используется для асинхронного ввода-вывода. В данном случае NULL, так как запись синхронная.

Возвращаемое значение:

Возвращает TRUE при успешной записи данных и FALSE при ошибке.

Что выполняет:

Записывает сообщение клиента в Mailslot, доступный на стороне сервера.

5. CloseHandle

Предназначение:

Закрывает дескриптор объекта, освобождая связанные с ним ресурсы.

Сигнатура:

```
BOOL CloseHandle(
HANDLE hObject // Дескриптор объекта, который нужно закрыть
);
```

Параметры:

hObject: Дескриптор, возвращенный функциями CreateMailslot или CreateFile.

Возвращаемое значение:

Возвращает TRUE, если дескриптор успешно закрыт, и FALSE, если произошла ошибка.

Что выполняет:

Завершает работу с дескриптором Mailslot или файлового объекта.

6. CreateMailslot

Предназначение:

Эта функция используется для создания нового Mailslot, который будет использоваться для отправки или получения сообщений. Она позволяет настроить параметры работы с Mailslot, например, максимальный размер сообщений.

Сигнатура:

```
HANDLE CreateMailslot(
LPCSTR lpName,            // Имя Mailslot
DWORD nMaxMessageSize,        // Максимальный размер одного сообщения
DWORD lReadTimeout,       // Время ожидания при чтении
LPSECURITY_ATTRIBUTES lpSecurityAttributes // Атрибуты безопасности (можно NULL)
);
```

Параметры:

lpName: Имя Mailslot, по которому другие процессы могут получить доступ к этому Mailslot.
nMaxMessageSize: Максимальный размер одного сообщения в байтах.
lReadTimeout: Время, через которое произойдет возврат ошибки при отсутствии сообщений.
lpSecurityAttributes: Атрибуты безопасности, обычно можно передать NULL, если атрибуты не требуются.

Возвращаемое значение:

Возвращает дескриптор Mailslot при успешном создании или INVALID_HANDLE_VALUE, если возникла ошибка.

Что выполняет:

Создает Mailslot с указанными параметрами.

После успешного создания Mailslot можно использовать его для отправки или получения сообщений с помощью WriteFile и ReadFile.

7. GetFileSize

Предназначение:

Функция используется для получения размера файла или устройства. В контексте Mailslot, эта функция может быть использована для проверки объема данных в Mailslot.

Сигнатура:

```
DWORD GetFileSize(
HANDLE hFile,       // Дескриптор файла или устройства
LPDWORD lpFileSizeHigh // Высокий 32-битный разряд размера
);
```

Параметры:

hFile: Дескриптор файла или устройства.
lpFileSizeHigh: Указатель на переменную, которая будет содержать старший 32-битный разряд размера файла (если размер превышает 4 ГБ).

Возвращаемое значение:

Возвращает размер файла в байтах. Если файл слишком большой, старший разряд возвращается через параметр lpFileSizeHigh.

Что выполняет:

Позволяет узнать размер файла или Mailslot, что может быть полезно для мониторинга объема данных в процессе передачи сообщений.

8. SetFilePointer

Предназначение:

Эта функция используется для перемещения указателя позиции в файле или устройстве. В контексте работы с Mailslot она может использоваться для управления текущей позицией записи или чтения в Mailslot, особенно если работа ведется с большими объемами данных.

Сигнатура:

```
DWORD SetFilePointer(
HANDLE hFile,          // Дескриптор файла или устройства
LONG lDistanceToMove,  // Количество байт для перемещения
PLONG lpDistanceToMoveHigh, // Старший 32-битный разряд
DWORD dwMoveMethod     // Метод перемещения: начало, текущая позиция или конец
);
```

Параметры:

hFile: Дескриптор файла или устройства, например, дескриптор Mailslot.
lDistanceToMove: Количество байт для перемещения указателя.
lpDistanceToMoveHigh: Указатель на переменную для старшего 32-битного разряда при необходимости (если размер файла превышает 4 ГБ).
dwMoveMethod: Определяет точку отсчета для перемещения (например, начало файла, текущая позиция или конец).

Возвращаемое значение:

Возвращает новое смещение в файле относительно начала (если dwMoveMethod = FILE_BEGIN), текущей позиции (если dwMoveMethod = FILE_CURRENT), или конца файла (если dwMoveMethod = FILE_END).

Что выполняет:

Перемещает указатель позиции в Mailslot, позволяя вам управлять текущей позицией для записи или чтения. Однако в случае Mailslot, это может не иметь большого значения, если не используется возможность работы с асинхронным вводом/выводом или прямым доступом к данным.

9. GetLastError

Предназначение:

Эта функция используется для получения кода последней ошибки, которая произошла в процессе работы с операционными системами Windows.

Сигнатура:

```
DWORD GetLastError();
```

Параметры:

Нет.

Возвращаемое значение:

Возвращает код последней ошибки в виде значения типа DWORD, который можно интерпретировать с использованием документации для Windows API.

Что выполняет:

Позволяет определить, что именно пошло не так в случае неудачного выполнения операции, например, при создании Mailslot или записи/чтении данных.

Важно для отладки и диагностики.

10. CancelIo

Предназначение:

Эта функция используется для отмены асинхронных операций ввода/вывода. Если операция чтения или записи была запущена асинхронно и необходимо ее прервать, используется эта функция.

Сигнатура:

```
BOOL CancelIo(
HANDLE hFile   // Дескриптор файла или устройства
);
```

Параметры:

hFile: Дескриптор файла или устройства, на котором осуществляется асинхронный ввод/вывод.

Возвращаемое значение:

Возвращает TRUE при успешной отмене операции, FALSE в случае ошибки.

Что выполняет:

Останавливает выполнение асинхронной операции ввода/вывода, если она еще не завершена. Например, если данные еще не были прочитаны из Mailslot, можно отменить текущую операцию.

11. FlushFileBuffers

Предназначение:

Эта функция используется для сброса всех данных, которые были записаны в буфер, в файл или устройство, т.е. она заставляет операционную систему физически записать данные на диск.

Сигнатура:

```
BOOL FlushFileBuffers(
HANDLE hFile  // Дескриптор файла или устройства
);
```

Параметры:

hFile: Дескриптор файла или устройства, например, дескриптор Mailslot.

Возвращаемое значение:

Возвращает TRUE, если операция прошла успешно, или FALSE, если произошла ошибка.

Что выполняет:

Для записи данных в Mailslot, если используется буферизация, эта функция может быть полезна для того, чтобы гарантировать, что все данные были записаны.

Это полезно для завершения записи и обеспечения того, чтобы все данные были физически отправлены.

12. ReadFileEx

Предназначение:

Функция используется для асинхронного чтения данных из файла или устройства. Эта версия функции предназначена для обработки ввода/вывода с использованием механизмов асинхронной обработки.

Сигнатура:

```
BOOL ReadFileEx(
HANDLE hFile,                    // Дескриптор файла или устройства
LPVOID lpBuffer,                 // Буфер для данных
DWORD nNumberOfBytesToRead,      // Количество байт для чтения
LPOVERLAPPED lpOverlapped,       // Указатель на структуру Overlapped для асинхронного чтения
LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine // Функция, которая будет вызвана по завершении операции
);
```

Параметры:

hFile: Дескриптор файла или устройства.
lpBuffer: Буфер для получения данных.
nNumberOfBytesToRead: Количество байт, которые нужно прочитать.
lpOverlapped: Указатель на структуру OVERLAPPED, которая используется для асинхронных операций.
lpCompletionRoutine: Указатель на функцию, которая будет вызвана после завершения операции.

Возвращаемое значение:

Возвращает TRUE, если операция была успешно запущена, и FALSE в случае ошибки (при этом функция асинхронно завершится). Для получения подробностей о завершении операции необходимо обработать структуру OVERLAPPED.

Что выполняет:

Это асинхронная версия функции ReadFile, которая позволяет выполнять операции чтения без блокировки потока выполнения программы.

Обычно используется для реализации более сложных сценариев, например, для серверных приложений, работающих с несколькими клиентами.

13. WriteFileEx

Предназначение:

Подобно функции ReadFileEx, эта функция используется для асинхронной записи данных в файл или устройство, что позволяет избежать блокировки потока и продолжать выполнение программы.

Сигнатура:

```
BOOL WriteFileEx(
HANDLE hFile,                    // Дескриптор файла или устройства
LPCVOID lpBuffer,                // Буфер с данными
DWORD nNumberOfBytesToWrite,     // Количество байт для записи
LPOVERLAPPED lpOverlapped,       // Структура для асинхронной записи
LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine // Функция, которая будет вызвана по завершении операции
);
```

Параметры:

hFile: Дескриптор файла или устройства.
lpBuffer: Буфер с данными, которые необходимо записать.
nNumberOfBytesToWrite: Количество байт для записи.
lpOverlapped: Структура для асинхронной записи.
lpCompletionRoutine: Функция, которая будет вызвана по завершении операции.

Возвращаемое значение:

Возвращает TRUE, если операция была успешно запущена.

Что выполняет:

Асинхронно записывает данные в файл или Mailslot, позволяя серверу не блокировать выполнение других операций.

Позволяет выполнять операции записи параллельно с другими задачами, что повышает производительность при обработке сообщений.