Межпроцессное взаимодействие

Передача информации

Сообщение WM\_COPYDATA

**Отправитель:**

COPYDATASTRUCT cds;

cds.cbData = (DWORD) nSize;

cds.lpData = (PVOID) pBuffer;

SendMessage (hWndTarget, WM\_COPYDATA, (WPARAM) hWnd, (LPARAM) &cds);

**Получатель:**

PCOPYDATASTRUCT pcds = (PCOPYDATASTRUCT) lParam;

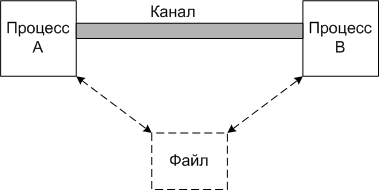
PBYTE pBuffer = (PBYTE) pcds -> lpData;

* Сообщение WM\_COPYDATA позволяет приложениям копировать данные между их адресными пространствами. При этом приложения не обязательно должны быть 32-разрядными — для 16-разрядных приложений поддерживается автоматическая трансляция указателей.
* Перед отправкой сообщения WM\_COPYDATA необходимо инициализировать структуру COPYDATASTRUCT с информацией о предстоящей пересылке данных, в том числе с указателем на блок данных. Затем с помощью функции SendMessage сообщение WM\_COPYDATA пересылается в принимающую программу; при этом параметр wParam содержит дескриптор окна вашей программы, а lParam - адрес структуры COPYDATASTRUCT.
* Когда сообщение поступает обработчику WM\_COPYDATA принимающей программы, его средствами указатель может быть скопирован из структуры COPYDATASTRUCT и использован, как любой другой указатель.
* В принимающем процессе размер блока данных, адрес которого содержит lpData, извлекается из элемента cbData. Значение, считываемое из элемента lpData принимающей программой, возможно, будет отличаться от значения, помещенного туда отправляющей программой. Этого следовало ожидать, поскольку перед передачей сообщения WM\_COPYDATA операционная система Windows NT выделяет в адресном пространстве принимающего процесса блок памяти, копирует данные из отправляющего процесса и обновляет значение lpData. По всей вероятности, в адресных пространствах этих процессов адреса размещении блока не совпадут.
* В структуре COPYDATASTRUCT имеется третье, необязательное поле, dwData, в котором можно передать 32-разрядное значение, определяемое в программе. Только не передавайте в этом поле указатель, потому что в принимающем процессе он не будет воспринят.
* При использовании метода WM\_COPYDATA необходимо помнить об одной детали: сообщения должны именно пересылаться, а не просто регистрироваться. Для того чтобы освободить память, выделенную в адресном пространстве принимающего процесса, операционная система должна быть информирована о моменте завершении пересылки. Кроме того, получатель сообщения WM\_COPYDATA должен обращаться с блоком данных так, словно он предназначен только для чтения. Чтобы внести изменении в полученные данные, в принимающей программе необходимо подготовить их локальную копию. И наконец, не следует сохранять указатель, переданный в элементе lpData, дл применения его в дальнейшем.

Анонимные каналы

* Анонимные каналы не имеют имен.
* Не пригодны для обмена через сеть.
* Главная цель – служить каналом между родительским и дочерним процессом или между дочерними процессами.
* Односторонний обмен.
* Не возможен асинхронный обмен.

Каналы



Канал представляет собой псевдофайл с органзацией типа буфера FIFO (first input and first output - первый вошел, первый вышел).

Образно говоря, канал представляет собой трубу (pipe) с двумя открытыми концами, в который один процесс пишет, а другой читает

Использование анонимных каналов

* Главная цель – служить каналом между родительским и дочерним процессом или между дочерними процессами.
* Родительский пpоцесс может быть консольным или GUI-пpиложение, но дочернее приложение должно быть консольным. Как вы знаете, консольное приложение использует стандартные дескрипторы для ввода и вывода.
* Если мы хотите пеpенапpавить ввод/вывод консольного приложения, мы можем заменить один дескриптор другим дескриптором одного конца канала. Консольное приложение не будет знать, что оно использует один конец канала. Оно будет считать, что это стандартный дескриптор. Это вид полимоpфизма на ООП-жаpгоне.
* Это мощный подход, так как нам не нужно модифицировать родительский процесс ни каким образом.

Создание анонимных каналов

BOOL CreatePipe(

PHANDLE *hReadPipe*,

PHANDLE *hWritePipe*, LPSECURITY\_ATTRIBUTES *lpPipeAttributes*,

DWORD *nSize*

);

ReadFile

WriteFile

* pReadHandle - это указатель на переменную типа dword, которая получит дескриптов конца чтения канала.
* pWriteHandle - это указатель на переменную типа dword, которая получить дескриптов конца записи канала.
* pPipeAttributes указывает на стpуктуpу SECURITY\_ATTRIBUTES, котоpая опpеделяет, наследуется ли каждый из концов дочерним процессом.
* nBufferSize - это предполагаемый pазмеp буфера, который канал заpезеpвиpует для использования. Это всего лишь предполагаемый pазмеp. Вы можете пеpедать NULL, чтобы указать функции использовать pазмеp по умолчанию.
* Не забудьте установить паpаметp bInheritable стpуктуpы SECURITY\_ATTRIBUTES в TRUE, чтобы дескриптры могли наследоваться.

Передача дескрипторов

* Установить паpаметp bInheritable стpуктуpы SECURITY\_ATTRIBUTES в TRUE, чтобы дескрипторы могли наследоваться.
* Вызов функции CreateProcess с параметром blnheritHandles = TRUE
* Передача дескрипторов (командная строка, сообщения…)
* Вызов функции DuplicateHandle

Дубликаты дескрипторов

BOOL DuplicateHandle(

HANDLE *hSourceProcessHandle*,

HANDLE *hSourceHandle*,

HANDLE *hTargetProcessHandle*,

LPHANDLE *lpTargetHandle*,

DWORD *dwDesiredAccess*,

BOOL *bInheritHandle*,

DWORD *dwOptions*

);

* Первый и третий параметры функции DuplicateHandle представляют собой описатели объектов ядра, специфичные для вызывающего процесса Кроме того, эти параметры должны идентифицировать именно процессы — функция завершится с ошибкой, если Вы передадите описатели на объекты ядра любого другого типа.
* Второй параметр, hSourceHandle, — описатель объекта ядра любого типа. Однако его значение специфично не для процесса, вызывающего DuplicateHandle, а для того, на который указывает описатель hSourceProcessHandie. Параметр phTargetHandle — это адрес переменной типа HANDLE, в которой возвращается индекс записи с копией описателя из процесса-источника. Значение возвращаемого описателя специфично для процесса, определяемого параметром phTargetProcessHandle.
* Предпоследние два параметра DuplicateHandle позволяют задать маску доступа и флаг наследования, устанавливаемые для данного описателя в процессе-приемнике. И, наконец, параметр dwOptions может быть 0 или любой комбинацией двух флагов. DUPLICATE\_SAME\_ACCESS и DUPLICATE\_CLOSE\_SOURCE
* Первый флаг подсказывает DuplicateHandle: у описателя, получаемого процессом-приемником, должна быть та же маска доступа, что и у описателя в процессе-источнике Этот флаг заставляет DuplicateHandle игнорировать параметр dwDesiredAccess.
* Второй флаг приводит к закрытию описателя в процессе-источнике. Он позволяет процессам обмениваться объектом ядра как эстафетной палочкой При этом счетчик объекта не меняется.

Пример использования анонимного канала

* Создаем анонимный канал с помощью CreatePipe.
* Теперь мы должны подготовить параметры, которые передадим CreateProcess (мы используем эту функцию для загрузки консольного приложения).
* Вызоваем CreateProcess, чтобы загрузить дочернее приложение. После того, как вызов прошел успешно, дочерний процесс все еще находится в спящем состоянии. Он загружается в память, но не запускается немедленно.
* Закройте дескриптор записи канала. Это необходимо, так как родительскому процессу нет нужды использовать этот дескриптор, а канал не будет работать, если открыть более чем один дескриптор записи.
* Теперь вы можете читать данные с помощью ReadFile. Вы должны последовательно вызывать ReadFile, пока она не возвратит ноль, что будет означать, что больше данных нет.
* Закроем дескриптор чтения канала.

NPFS (Named Pipe File System)

* **Named Pipe File System** является виртуальной файловой системой, которая управляет каналами **named pipes**.
* Каналы named pipes относятся к классу файловых объектов (API Win32).
* RPC реализован как надстройка над NPFS;
* Канал представляет собой виртуальное соединение, по которому передается информация от одного процесса к другому.
* Канал может быть однонаправленным или двунаправленным (дуплексным).

Работа с именованными каналами

* Серверный процесс создает канал на локальном компьютере с помощью функции программного интерфейса Win32 **"CreateNamedPipe"**.
* Серверный процесс активизирует канал при помощи функции **"ConnectNamedPipe"**, после чего к каналу могут подключаться клиенты.
* Далее производится подключение к каналу \\computer\_name\pipe\pipe\_name посредством вызова функции **"Create File"**.

Создание именованного канала

**HANDLE CreateNamedPipe** (  
**LPCTSTR** lpName,   
**DWORD** dwOpenMode,   
**DWORD** dwPipeMode,   
**DWORD** nMaxInstances,   
**DWORD** nOutBufferSize,   
**DWORD** nInBufferSize,   
**DWORD** nDefaultTimeOut,   
**LPSECURITY\_ATTRIBUTES** lpSecurityAttributes

);

Параметры создания канала

* lpName – имя именованного канала;
* dwOpenMode – определяет направление передачи, возможные варианты - **PIPE\_ACCESS\_DUPLEX**, **PIPE\_ACCESS\_INBOUND**, **PIPE\_ACCESS\_OUTBOUND** ;
* dwPipeMode – способ передачи информации (**PIPE\_TYPE\_BYTE** или **PIPE\_TYPE\_MESSAGE**
* nMaxInstances – количество каналов с данным именем которые может открыть пользователь;
* nOutBufferSize и nInBufferSize – размер буферов приема и отправки;
* nDefaultTimeout – максимальное время ожидания при асинхронном вводе/выводе через канал;
* lpSecurityAttributes – указатель на структуру **SECURITY\_ATTRIBUTES**, которая задает уровень защиты создаваемого объекта.

Подключение к именованному каналу

BOOL ConnectNamedPipe (  
HANDLE hNamedPipe,  
LPOVERLAPPED lpOverlapped );

BOOL DisconnectNamedPipe (  
HANDLE hNamedPipe

);

* После того как канал создан, сервер подключается к нему с помощью функции **ConnectNamedPipe** () и начинает ожидать подключения клиента.
* Необходимо отметить, что подключение сервера к каналу может осуществляться как синхронным, так и асинхронным способом. В первом случае ConnectNamePipe возвращает управление программе лишь после того, как клиент подключился к каналу, во втором же случае, возврат управления происходит сразу же, а уведомление программы о подключении осуществляется через структуру **OVERLAPPED** , указатель на которую передается вторым параметром в ConnectNamedPipe.
* После завершения обмена необходимо отключиться от канала с помощью функции **DisconnectNamedPipe**.
* Затем можно снова открыть канал и ожидать подключения следующего клиента, а по завершению работы с каналом необходимо закрыть его дескриптор функцией **CloseHandle** ().

Обмен данными по именованному каналу

**BOOL ReadFile/WriteFile** (  
**HANDLE** hFile,   
**LPVOID** lpBuffer,   
**DWORD** nNumberOfBytesToRead,   
**LPDWORD** lpNumberOfBytesRead,  
**LPOVERLAPPED** lpOverlapped

);

Работа с каналом и ее завершение

* После установления виртуального соединение серверный процесс и клиентский процесс могут обмениваться информацией при помощи пар функций **"ReadFile"** и **"WriteFile"**.
* При помощи одного и того же канала сервер может одновременно обслуживать нескольких клиентов. Для этого серверный процесс может создать N-ное количество экземпляров канала, вызвав N-ное количество раз функцию **"CreateNamedPipe"** (при этом в каждом вызове должно быть указано одно и то же имя канала).
* Клиентский процесс может отключиться от канала в любой момент с помощью функции **"CloseHandle"**. Серверный процесс может отключить клиента в любой момент с помощью функции **"DisconnectNamedPipe"**.

Почтовые ящики (MailSlots)

* Почтовые слоты — способ однонаправленной коммуникации. Приложение-сервер открывает слот, а клиенты могут писать в него. Слот сохраняет сообщения до тех пор, пока сервер их не прочтет. Разумеется, одно приложение может одновременно быть сервером и клиентом, обеспечивая двунаправленную связь. При этом приложения могут находиться даже на разных компьютерах в сети.
* Процесс-сервер может завести почтовый ящик и дать ему имя, глобальное в сети. Любой клиент может с помощью операций работы с файлами отправить данные в этот ящик. Сервер, по мере необходимости, может читать переданные ему данные. Кроме этого, возможно широковещательная передача информации клиентом всем серверам домена.

*Mailslot* является одним из механизмов, предназначенных для осуществления обмена данными между процессами (IPC). При этом процессы могут быть запущены как на одной ПЭВМ (локально), так и разных ПЭВМ, включённых в одну ЛВС (удалённо).

* *Mailslot* представляет собой псевдофайл, хранящийся в памяти. Для доступа к данным, содержащимся в этом псевдофайле, используются стандартные файловые функции Win32.
* Объект *Mailslot* является временным объектом. После того, как будет закрыт последний дескриптор, ссылающийся на объект *Mailslot*, сам объект с данными будет уничтожен.
* Обмен данными посредством *Mailslot* осуществляется в большинстве случаев между двумя процессами – *клиентом* и *сервером*.

Формат имени сервера

* При создании объекта *Mailslot* сервером, имя объекта должно иметь следующий формат: *\\.\mailslot\[path]name* Имя объекта *Mailslot* должно содержать две наклонные черты влево, точку, ещё одну наклонную черту влево, слово «*mailslot*» и последнюю наклонную черту. После последней наклонной черты указывается собственно имя создаваемого объекта. Имя может также содержать путь. Примеры имени создаваемого объекта *Mailslot*:
* *\\.\mailslot\Test.msl \\.\mailslot\SampleDir\Sample*

Форматы имени клиента

* Для того чтобы записать сообщение в *Mailslot*, клиент обращается к нему по имени. При этом если клиент и сервер запушены на одной ПЭВМ, то формат имени, используемый клиентом, может совпадать с форматом имени сервера. Однако чаще необходимо записывать сообщения в удалённый *Mailslot*, для чего необходимо использовать следующий формат имени объекта *Mailslot*: *\\ComputerName\mailslot\Test.msl* Здесь *ComputerName* – сетевое имя ПЭВМ, на которой расположен сервер *Mailslot*.
* Для того чтобы клиент мог поместить сообщение в каждый *Mailslot* с данным именем, созданный в пределах домена, формат имени объекта *Mailslot* для клиента должен быть следующим: *\\DomainName\mailslot\Test.msl* Здесь *DomainName* – имя домена, включающие в себя те ПЭВМ, на которых запущены серверы *Mailslot*. Для того чтобы клиент мог поместить сообщение в каждый *Mailslot* с данным именем в первичном системном домене, имя объекта *Mailslot* должно иметь следующую форму: *\\\*\mailslot\Test.msl*

Клиенты, сервера и имена

* **MailSlot** cервер – является процессом, который создает и, обладает **MailSlot**. Когда сервер создает **MailSlot**, он получает указатель. Этот указатель должен использоваться, когда процесс читает сообщения от **MailSlot**. Только процесс, который создает **MailSlot** или получил указатель некоторым другим механизмом может прочитать данные из **MailSlot**. Все **MailSlot** локальные на процессе, который создает их; процесс не может создать дистанционный **MailSlot**.
* **MailSlot** Клиент – является процессом, который пишет сообщение в **MailSlot**. Любой процесс, который имеет имя **MailSlot** может записать в него информацию.

Есть два вида программ использующих данную возможность: MailSlot сервер MailSlot клиент.

Создание почтового ящика на сервере

HANDLE CreateMailslot (

LPCTSTR lpName, *// имя*

DWORD nMaxMessageSize, *// максимальный //размер*

DWORD lReadTimeout, *// интервал-тайм аута //чтения*

LPSECURITY\_ATTRIBUTES lpSecurityAttributes

*// информация о безопасности*

);

* Для открытия канала, созданного на другом компьютере в сети, необходимо указать имя в формате
* \\ИмяКомпьютера\mailslot\[Путь]ИмяКанала
* Можно открыть канал для передачи информации сразу всем компьютерам указанного домена. Для этого формируется имя
* \\ИмяДомена\mailslot\[Путь]ИмяКанала
* Для передачи сообщения всем компьютерам первичного домена имя задается в форме
* \\\*\mailslot\[Путь]ИмяКанала

Пример создания сервера

HANDLE hSlot = NULL;

hSlot = CreateMailslot ("\\\\computername\\mailslot\\messngr", 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hSlot != INVALID\_HANDLE\_VALUE)

{  
char buffer[255]; DWORD nBytesRead;

ReadFile(hSlot, &buffer, 255, &nBytesRead, NULL);

…

}

Создание клиента почтового ящика

HANDLE hSlot = CreateFile(("\\\\computername\\mailslot\\messngr", GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  
OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hSlot != INVALID\_HANDLE\_VALUE)

{  
char buf = "From\0\To\0Message\0";  
uint cb = sizeof(buf);  
WriteFile(hSlot, buf, cb, &cb, NULL);

…

}

Использование mailslot

* Использование мэйлслотов особенно удобно в системах такого pода, pаботающих в пpеделах локальной сети. Пpогpамма-сеpвеp создает на своем компьютере мэйлслот с именем, известным всем клиентам (напpимеp BWCronServerMailSlot). Пpогpамма-клиент производит соединение с сеpвеpом путем создания клиентского мэйлслота (BWCronClientMailSlot) и пpовеpки существования сеpвеpа. Такую пpовеpку можно произвести путем посылки сеpвеpу контрольного сообщения. Сеpвеp дает добро на подключение либо отклоняет запрос. Дальнейший обмен данными будет производиться путем посылки сеpвеpу сообщений, и получения ответов от сеpвеpа через клиентский мэйлслот.
* Мэйлслоты кроме систем "клиент-сеpвеp" можно использовать, напpимеp, для определения, запущена ли еще одна копия пpогpаммы где-либо в локальной сети. Это делается посылкой сообщения всем компьютерам в заданном домене (второй сеpвеp прикидывается клиентом и пытается установить связь с сеpвеpом. Если связь установлена, то работу не продолжаем, а если нет, то можно самому работать сеpвеpом).

В **MSDN** написано, что если клиент открывает слот прежде чем слот был создан сервером, то он получит **INVALID\_HANDLE\_VALUE**

Получение информации о почтовом ящике

BOOL GetMailslotInfo (

HANDLE hMailslot, *// указатель на слот*

LPDWORD lpMaxMessageSize, *// максимальный размер*

LPDWORD lpNextSize, *// размер следующего*

LPDWORD lpMessageCount, *// количество сообщений*

LPDWORD lpReadTimeout *// тайм аут*

);

Изменение настроек почтового ящика

BOOL SetMailslotInfo( HANDLE *hMailslot*, DWORD *lReadTimeout* );