Межпроцессное взаимодействие

Как уже упоминалось ранее, процесс - один из основных объектов в любой операционной системе. Собственно, сама ОС и существует для того, чтобы позволить пользователю (в широком смысле) выполнять процессы, решающие те или иные задачи. Поэтому в течение своей работы процессы неизбежно тем или иным образом взаимодействуют с ядром ОС. Однако абсолютное большинство процессов не являются изолированными - зачастую возникает необходимость "побеседовать" с "коллегами". В современных ОС даже многие подсистемы самой ОС могут быть реализованы как процессы - примером служит OS Windows, где это применяется повсеместно - скажем, подсистема времени выполнения клиент/сервер (Client/Server Runtime Subsystem), отвечающая за ввод/вывод в консоли и некоторые другие функции, вынесена в отдельный процесс `csrss`. Ну и конечно прикладные программы также могут иметь необходимость обмениваться данными между собой. Для этой цели современными ОС предусматривается большое количество разнообразных методов, например:

- файлы;
- сокеты;
- сигналы;
- каналы (pipe), именованные и нет;
- почтовые ячейки (mail slot);
- сообщения (разного рода);
- RPC;
- разделяемая (общая) память.

Более подробно общие концепции различных механизмов взаимодействия будут рассмотрены на лекционных занятиях; здесь же мы сконцентрируемся на практических примерах наиболее часто используемых механизмов межпроцессного взаимодействия.

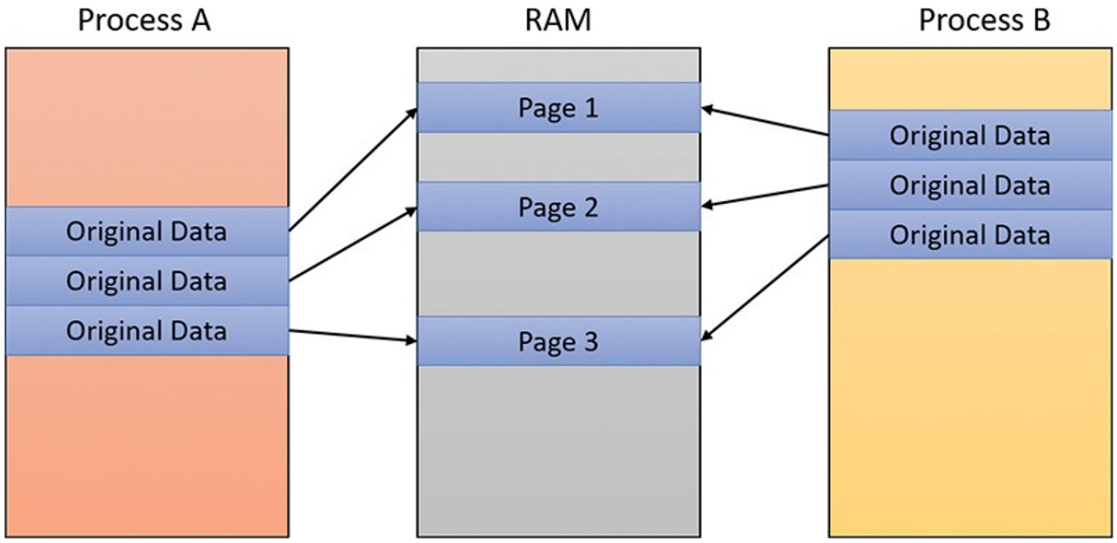
Механизмы межпроцессного взаимодействия в Windows NT

Windows NT, будучи современной операционной системой, поддерживает все вышеперечисленные способы межпроцессного взаимодействия, а также реализует несколько своих, специфичных только для данного семейства ОС, механизмов (DDE, OLE), которые в данном курсе не рассматриваются - во-первых, из-за своей специфичности, а во-вторых, из-за того, что они достаточно высокоуровневы и все равно так или иначе работают поверх других, более простых механизмов. Но даже в реализациях тех механизмов, которые присутствуют в ОС Windows, имеются определенные особенности. Так, процесс может послать сигнал только самому себе, что делает применение сигналов для межпроцессной коммуникации невозможным. Об особенностях каждого из прочих механизмов будет написано в соответствующем разделе.

Разделяемая память

Разделяемая память позволяет группе из двух и более процессов иметь участок памяти, операции над которым будут приводить к его изменению в адресных пространствах

всех процессов группы. Задачу поддержания идентичности ОС берет на себя, но вот проблема взаимного исключения остается целиком и полностью на совести программиста - это будет рассмотрено в следующих частях курса.



описание

В Windows для создания разделенного участка памяти используется уже знакомая нам функция CreateFileMapping. Первым параметром ей передается специальное значение `INVALID_HANDLE_VALUE`, говорящее системе о том, что данное отображение не должно быть привязано к какому-либо файлу (на самом деле, "под капотом" оно будет отнесено к некоторому участку в файле подкачки, но это уже внутренние детали архитектуры подсистемы виртуальной памяти ОС Windows, не столь существенные на данном этапе). Но если мы не привязываем отображение к какому-либо файлу, как нам тогда в разных процессах получить отображение одних и тех же данных? Для этого служит последний параметр функции, позволяющий указать **имя** отображения. Два одноименных отображения, созданные в разных процессах, будут иметь под собой один и тот же участок памяти. Таким образом несколько процессов могут отобразить в свое адресное пространство одни и те же данные.

Дальнейшая работа с полученным отображением происходит ровно так же, как и с обычным отображением файла на память:

- При помощи MapViewOfFile создается окно просмотра, с которым можно обращаться, как с линейным массивом байтов.
- Над данными в окне просмотра производятся необходимые операции.
- Окно просмотра уничтожается вызовом UnmapViewOfFile.
- Уничтожается само отображение с помощью CloseHandle.

Почтовые ячейки

Почтовые ячейки (mail slot) предоставляют механизм одностороннего взаимодействия между процессами. Один процесс (сервер почтовой ячейки) создает ячейку (иногда, продолжая аналогию с почтой, ее называют почтовым ящиком), после чего все прочие процессы (клиенты) получают возможность отправлять ему сообщения, "складывая" их

в эту почтовую ячейку. Двусторонняя коммуникация при помощи почтовых ячеек также возможна: при этом каждый из процессов создаст себе по почтовому ящику, становясь сервером, и одновременно выступает клиентом по отношению к другим процессам. Более того, в Windows существует возможность передачи сообщений при помощи широковещательной рассылки и даже по сети в пределах домена (однако по сети они передаются при помощи датаграмм - иными словами, доставка сообщения не гарантируется).

Как правило, размер сообщения, передаваемого при помощи механизма почтовых ячеек, ограничен сверху. Например, в Windows он задается при создании почтовой ячейки сервером, а при широковещательной отправке задан фиксированно и не может превышать 400 байтов. Попытка отправить сообщение размером свыше лимита приведет к неудаче.

Механизм почтовых ячеек, ввиду своей простоты, возможности ограничения верхнего предела размера сообщений и задания таймаута отправки/получения является широко распространенным механизмом межпроцессного взаимодействия в микроядерных ОС и особенно - в ОС реального времени (RTOS).

В Windows с точки зрения клиента почтовая ячейка ничем не отличается от обычного файла, за исключением того, что в нее возможна исключительно запись блоками ограниченного размера, и имени, которое выглядит как `"\\.\mailslot\sample_mailslot"`, где `sample_mailslot` - имя почтовой ячейки, а точка - указание на то, что ячейка находится на текущем компьютере. Для доступа к почтовой ячейке, находящейся на другом ПК, можно здесь указать вместо точки имя компьютера, на котором находится интересующая нас ячейка. Работа с ней ведется при помощи уже знакомых функций CreateFile, WriteFile, CloseFile.

На сервере же все немного интереснее. Почтовая ячейка создается вызовом `CreateMailslot` и опрашивается на наличие в ней сообщений процедурой `GetMailslotInfo`. Чтение производится при помощи `ReadFile`.

CreateMailslot

Создает новую почтовую ячейку.

Параметры:

- Имя почтовой ячейки, по которому она будет доступна.
- Максимальный размер сообщения, который сможет принять ячейка.
- Таймаут ожидания чтения сообщения, то есть время, которое функция чтения сообщения из ячейки может подождать, если в ячейке отсутствуют сообщения.
- Атрибуты безопасности.

Созданная почтовая ячейка уничтожается вызовом `CloseHandle`.

GetMailslotInfo

Позволяет получить некоторую информацию о почтовой ячейке. В параметрах процедуры передаются указатели на переменные, в которых данная информация будет сохранена.

Параметры:

- Хэндл почтовой ячейки.
- Указатель на переменную, где будет сохранен максимальный размер сообщения, принимаемый данной почтовой ячейкой.
- Указатель на переменную, где будет сохранен размер первого ожидающего приема сообщения, или -1, если сообщений нет.
- Указатель на переменную, где будет сохранено количество сообщений, ожидающих приема.
- Указатель на переменную, где будет сохранен таймаут ожидания чтения сообщения.

SetMailslotInfo

Позволяет указать значение таймаута ожидания чтения сообщения.

Параметры:

- Хэндл почтовой ячейки.
- Новое значение таймаута ожидания чтения сообщения.

Каналы

Для организации общения между *строго двумя* не обязательно локальными процессами можно воспользоваться механизмом каналов. Канал (от англ. *Pipe*, не путать с channel!) в Windows по своей сути есть фрагмент разделяемой памяти, в который можно писать и читать данные, при этом канал может быть как однонаправленным (*симплексным*: процесс-сервер пишет данные, процесс-клиент - читает), так и двунаправленным (*дуплексным*). Описание канала представляет собой два дескриптора: дескриптор чтения и дескриптор записи, таким образом, разработчик сам решает, какая операция позволена серверу и клиентам.

Одна из главных особенностей каналов в WinAPI заключается в том, что после их создания работа с ними, как и в случае почтовых ячеек, ведётся хорошо знакомыми функциями `ReadFile`, `WriteFile` и т.д.

Существует 2 типа каналов:

Неименованный канал

Пригоден для взаимодействия уже некторым образом связанных между собой локальных процессов (как правило, применяется для связи между родительским процессом и его потомками). Утверждается, что обладает меньшими накладными расходами. Не поддерживает использование асинхронных операций чтения/записи. Читать и писать в неименованный канал могут сразу несколько процессов, причем порядок, в котором будут производиться операции, заранее не определен и лежит целиком на совести программиста.

CreatePipe

Создаёт неименованный канал указанного размера и получает его дескрипторы чтения и записи.

Параметры:

- Указатель на переменную, в которую будет размещён дескриптор чтения.
- Аналогично, только для дескриптора записи в канал.
- Атрибуты безопасности (или `NULL` если не нужны).
- Размер буфера канала в байтах. Можно указать `0` для использования значения по умолчанию.

Возвращает `TRUE` при успехе, `FALSE` при неудаче.

Именованный канал

Основная особенность - взаимодействие с помощью именованных каналов может выходить за пределы локальной машины. Для этого используется аналогичный почтовым ячейкам механизм адресации каналов вида `\\.\pipe\sample_pipe` где `.` - адрес машины в сети (в данном случае - текущая машина), `sample_pipe` - название канала на этой машине. Кроме того, в отличие от неименованных каналов, при открытии именованного канала клиентом для него создается свой собственный объект, описывающий канал. Таким образом, операции записи от разных клиентов изолированы друг от друга.

CreateNamedPipe

Параметры:

- Название канала.
- Режим открытия канала (дуплексный, однонаправленный приём, однонаправленная запись - см. MSDN).
- Режим работы канала (чтение\запись побайтово\по сообщениям, блокирующий\неблокирующий режим и т.д.).
- Максимальное количество экземпляров канала.
- Размер исходящего буфера.
- Размер входящего буфера.
- Атрибуты безопасности.

Возвращает "серверный" дескриптор канала или `INVALID_HANDLE_VALUE`.

CallNamedPipe

Клиентская функция подключения к существующему именованному каналу.

Параметры:

- Имя.
- Указатель на буфер, из которого следует отправить данные в канал.
- Размер этого буфера.

- Указатель на буфер-приёмник.
- Размер этого буфера.
- Указатель на переменную, куда будет записано количество прочитанных байтов из канала.
- Таймаут в миллисекундах.

Альтернативно можно подключаться к каналу с помощью `CreateFile`, просто передав имя канала вместо имени файла.

Сообщения оконной подсистемы

С ними мы уже знакомы - при помощи них ОС сообщает окну о событиях клавиатуры, мыши, изменения размера окна и т.п. Однако существует возможность использовать свой собственный тип сообщения с `wParam` и `lParam` для личных нужд. Здесь возможны два варианта:

- Если необходима коммуникация исключительно в пределах одного приложения между различными окнами одного и того же класса, то можно выбрать произвольным образом идентификатор, значение которого будет превосходить `WM_USER` и использовать его. Однако, такой способ не подходит для взаимодействия между окнами различных приложений.
- Если все же необходимо взаимодействие нескольких приложений (или нескольких окон различных классов), то требуется зарегистрировать в системе новый класс сообщения. Для этого используется процедура `RegisterWindowMessage`.

Для отправки сообщения в обоих случаях используется стандартные процедуры `SendMessage` и `PostMessage`, а прием осуществляется абсолютно точно так же, как и всех прочих сообщений - в функции-обработчике.

RegisterWindowMessage

Глобально регистрирует новый тип сообщения.

Параметры:

- Имя типа сообщения. При вызове процедуры с одним и тем же именем в различных приложениях будет возвращен один и тот же идентификатор сообщения, что позволяет приложениям обмениваться данными.

Возвращаемое значение:

- Идентификатор нового типа сообщения или 0, если произошла ошибка.

PostMessage

Помещает новое сообщение в очередь для указанного окна, реализуя асинхронную отправку сообщения.

Параметры:

- Дескриптор окна.
- Тип сообщения.

- Значение `wParam`.
- Значение `lParam`.

SendMessage

Отправляет сообщение указанного типа с указанными параметрами указанному окну. Разница с `PostMessage` в том, что данная функция **синхронная**, то есть она вызывает обработчик сообщений указанного окна и ждёт его завершения - и всё это мимо очереди сообщений!

Параметры:

- Дескриптор окна или `HWND_BROADCAST` для рассылки вообще всем окнам верхнего уровня в системе.
- Тип сообщения.
- Значение `wParam`.
- Значение `lParam`.

Полезные ссылки

- Механизмы межпроцессного взаимодействия в ОС Windows https://docs.microsoft.com/en-us/windows/desktop/ipc/interprocess-communications