Лабораторная работа № 2

Межпроцессное взаимодействие

# Указания к выполнению лабораторной работы

## Средства Windows API для работы с файлами и объектами

### Описатели объектов (object handles)

Устройство внутренних структур ОС весьма сложно. Например, с открытым файлом связана информация о его расположении в файловой системе, о количестве считанных байт, служебные буферы для ускорения работы и т. п. Эти подробности не нужны прикладному программисту и не должны быть ему доступны, а требуется простой способ указать ОС на конкретный её внутренний объект. С этой целью широко применяются описатели объектов (object handles), называемые также дескрипторами (descriptor). С точки зрения прикладного программиста это простые переменные, обычно типа HANDLE (в Windows) или int (в \*nix), хотя используются и другие типы. Конкретные их значения не важны, главную роль играет то, что одно и то же значение соответствует одному и тому же объекту внутри ОС. Например, функция CreateFile() возвращает HANDLE открытого файла, который затем можно передать функции WriteFile() для записи в тот же файл. Специальное «некорректное» значение описателя — INVALID\_HANDLE\_VALUE, использовать NULL вместо него неправильно. Каждый полученный описатель должен быть по окончании работы с ним закрыт функцией CloseHandle().

### Работа с файлами

Функция CreateFile() используется для доступа к файлам, а также к некоторым другим объектам IPC. Она возвращает описатель открытого файла. Параметр lpFileName задает имя файла или объекта IPC. Параметр dwDesiredAccess — набор битовых флагов, обозначающий, какого рода доступ необходим (на чтение, на запись и т. п.). Параметр dwShareMode важен, если файл или объект используется одновременно несколькими приложениями (случай IPC), и указывает, какие операции допустимы при совместном использовании (чтение, запись и т. п.). Прочие параметры используются при работе с файловой системой и для IPC не важны (кроме lpSecurityAttributes, необязательных параметров безопасности).

Чтение и запись данных выполняется функцией ReadFile() и WriteFile() соответственно. Последний параметр обеих, lpOverlapped, применяется для обмена данными в асинхронном режиме; в данной работе это не используется.

## Средства межпроцессного взаимодействия

Не рассматривается механизм сигналов (signals), реализуемый только в POSIX-совместимых ОС, механизм оконных сообщений, присутствующий только в ОС Windows, а также способы синхронизации процессов и потоков, которым посвящена ЛР № 4.

### Отображаемые в память файлы (memory-mapped files)

[Отображение файлов в память](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366556) позволяет работать с содержимым файла или его частью как с простой областью памяти. Если одну и ту же область некоего файла отображают в собственную память несколько программ, изменения, вносимые любой из них в эту область, становятся немедленно доступны другим программам. Это можно использовать для взаимодействия между процессами (inter-process communication, IPC). Один процесс может записывать в область памяти сообщение (передавать данные), а другие процессы — считывать сообщение из памяти (получать данные).

### Именованные каналы (named pipes)

[Именованные каналы](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590) — это специальные объекты файловой системы для межпроцессного взаимодействия, обладающие следующими свойствами:

1. Последовательность доступа. Данные записываются и считываются по порядку, «перемотка» невозможна. Считываемые данные извлекаются (удаляются) из канала.
2. Направленность. Данные записываются с одного конца канала, а вычитываются с другого в порядке поступления. В Windows существуют т. н. дуплексные (двунаправленные) каналы, которые работают как пара каналов в противоположных направлениях.
3. Ограниченный объем записанных, но не считанных данных, находящихся в канале. Попытка поместить в канал больше данных приводит либо к ошибке, либо к ожиданию, пока место в канале освободится.

Одна программа, называемая в Windows сервером, создает именованный канал. Другие программы, называемые клиентами, подключаются к каналу, указывая его имя. В любой момент времени сервер может быть соединен только с одним клиентом (по одному каналу). Далее и сервер, и клиент могут работать с каналом как с файлом, учитывая названные выше ограничения (например, последовательность доступа). Любую порцию данных может вычитать только один клиент; сервер считывает все записанные данные от всех клиентов.

Каналы могут быть ориентированы на передачу байт или сообщений (только в Windows). В случае сообщений передача выполняется неделимыми порциями байт, и не нужно обрабатывать случаи, когда вычитана лишь часть сообщения; однако и размер сообщений ограничен 64 КБ. Кроме того, доступны специальные функции, упрощающие и ускоряющие передачу. Сообщения могут вычитываться и побайтово; обратное неверно.

### Очереди сообщений

Очередью сообщений (message queue, MQ) называется механизм, позволяющий одним процессам помещать сообщения в очередь, а другим — извлекать сообщения по одному в порядке добавления.

В ОС Windows механизм очередей сообщений реализуется т. н. почтовыми ящиками ([mailslots](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365576)), которые не следует путать с ящиками электронной почты (mailboxes). Процесс, создавший почтовый ящик функцией CreateMailslot(), считается *сервером.* Только он может получать сообщения из почтового ящика функцией ReadFile(). *Отправители,* или *клиенты,* — любые другие процессы — записывают сообщения в почтовый ящик функцией WriteFile(); сервер  же отправлять сообщения в созданный ящик не может. Количество сообщений в ящике позволяет определить функция GetMailslotInfo().

Сообщения не имеют адресатов и адресантов, то есть нельзя установить процесс-отправитель при получении (если этого не указать в самом сообщении). Почтовые ящики доступны по локальной сети, то есть через почтовый ящик можно передавать сообщения между процессами на разных машинах. Доставка сообщений при этом, однако, не гарантируется, а их размер ограничен.

### Перенаправление стандартных потоков через каналы

[Анонимные каналы](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365139) (anonymous pipes или просто pipes) работают идентично именованным, однако не имеют связанного с ними имени и пути в файловой системе. Как следствие, невозможно открыть анонимный канал, созданный другим процессом, так как без имени канала нельзя его идентифицировать. При создании анонимного канала функцией CreatePipe() процесс получает два описателя для обоих концов канала: чтения (read end) и записи (read end) — анонимный канал всегда однонаправленный.

Каждый процесс имеет три стандартных потока: ввода (standard input, «stdin»), вывода («stdout») и ошибок («stderr»). Для терминальных программ поток ввода ассоциируется с клавиатурой, а потоки вывода и ошибок — с экраном. При создании нового процесса функцией CreateProcess() можно указать в полях структуры STARTUPINFO, какие дескрипторы, полученные создающим (родительским) процессом, использовать в качестве стандартных потоков порождаемого (дочернего) процесса. Так, можно заменить стандартный поток дочернего процесса на конец анонимного канала.

Однако описатель, полученный одним процессом как конец канала, в общем случае нельзя использовать в другом как поток ввода-вывода. Изменить такое поведение можно, разрешив наследование данного описателя дочерним процессом при помощи функции SetHandleInformation(). Наследование описателей не связано с наследованием в объектно-ориентированном программировании, это термины-омонимы.

На рис. 1 показано, как родительский процесс может, создав анонимные каналы и перенаправив в них стандартные потоки дочернего процесса, взаимодействовать с последним. Важно не путать направленность каналов. На рис. 1 дочерний процесс получает конец для чтения в качестве стандартного потока ввода, поскольку будет считывать из него данные. Родительскому процессу остается конец для записи того же канала. Таким образом, канал служит потоком *вывода* для родительского процесса и потоком *ввода* для дочернего. При перенаправлении стандартного вывода и потока ошибок дочернего процесса ситуация прямо противоположная.

**Родительский процесс**

**Дочерний процесс**

SetHandleInformation(…, HANDLE\_FLAG\_INHERIT, HANDLE\_FLAG\_INHERIT) × 3

input\_write input\_read

GetStdHandle(STD\_INPUT\_HANDLE)

WriteFile(input\_write, …)

CreatePipe()

GetStdHandle(STD\_OUTPUT\_HANDLE)

output\_read output\_write

ReadFile(output\_read, …)

error\_read error\_write

GetStdHandle(STD\_ERROR\_HANDLE)

ReadFile(error\_read, …)

CreatePipe()

CreatePipe()

Рисунок 1 — Взаимодействие родительского процесса с дочерним   
через стандартные потоки и анонимные каналы

Примером программы, работающей описанным образом, хотя и в другой ОС, является сервер SSH (к которому подключается клиент PuTTY). Стандартные потоки запускаемых на удаленной машине программ перенаправляются; данные, полученные от клиента, передаются на их стандартный ввод, а считанные со стандартного вывода — пересылаются обратно по сети.

# Задание на лабораторную работу

Проверку работы программ вариантов 1—3 следует выполнять, запуская несколько экземпляров приложения, например, из «Проводника». В примерах **полужирным** начертанием обозначен ввод пользователя.

Вариант 1

Требуется написать программу для обмена текстовыми сообщениями между экземплярами через проецируемые в память файлы, которая:

1. Запрашивает у пользователя наименование разделяемой области памяти.
2. Пытается открыть указанную область функцией OpenFileMapping(); если области не существует, создает её функцией CreateFileMapping().

*Указание.* Отсутствие разделяемой области памяти можно диагностировать, если функция OpenFileMapping() завершилась ошибкой с соответствующим кодом (который можно получить функцией GetLastError()). Узнать нужный код ошибки можно по MSDN.

1. Проецирует участок разделяемой области памяти фиксированного размера.
2. Запрашивает у пользователя, что следует сделать:
   1. Записать строку в разделяемую область памяти. Необходимо предложить пользователю ввести строку, а затем записать в спроецированный участок разделяемой памяти.
   2. Считать строку в разделяемую область памяти. Необходимо вывести на экран содержимое спроецированного участка памяти как строку.
   3. Завершить работу. Необходимо закончить проецирование функцией UnmapViewOfFile() и отключиться от открытой или созданной разделяемой области памяти функцией CloseHandle().

*Указание.* Целесообразно взять за основу [официальный пример](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366551) работы с разделяемой областью памяти, соединив две представленные в нем программы в одну.

Проверить экспериментально и отразить результаты в отчете:

1. Возможность использования любой программы для обмена данными через файл, который был спроецирован в память.
2. Возможность работы со спроецированной областью файла после того, как завершилась создавшая эту область программа (из программ, которые открыли эту область еще при работе создавшей).

Пример организации тестирования с двумя процессами приведен ниже.

**Процесс № 1:**

Enter mapping name: **communication**  
Connecting to `communication'... failed.  
Creating new mapping `comminication'.  
Enter 1 for writing, 2 for reading, 3 for exit: **1**  
Enter the message: **Hello!**  
Enter 1 for writing, 2 for reading, 3 for exit:

**Процесс № 2:**

Enter mapping name: **communication**  
Connecting to `communication'... done.  
Enter 1 for writing, 2 for reading, 3 for exit: **2**  
The message is `Hello!'.

Вариант 2

Написать пару программ, взаимодействующих через именованные каналы. Программа-сервер хранит строки-значения по строкам-ключам. Программа-клиент, отправляя программе-серверу команды, добавляет, удаляет и получает значения.

### Программа-сервер:

1. Запрашивает у пользователя имя канала и создает дуплексный, ориентированный на сообщения канал с заданным именем функцией CreateNamedPipe().
2. Ожидает подключения клиента к каналу функцией ConnectNamedPipe().
3. Считывает из подключенного канала одну строку-команду, состоящую из имени и аргументов, разделенных пробелами (ни в имени, ни в аргументах пробелов нет) функцией ReadFile() и выполняет полученную команду:
   1. Сохранить значение по ключу. Формат команды:

set ключ значение

Необходимо сохранить в памяти значение под указанным ключом и записать в канал строку acknowledged.

* 1. Получить значение по ключу. Формат команды:

get ключ

Если ключ имеется в хранилище, следует записать в канал строку в формате

found значение

В противном случает следует записать в канал строку missing.

*Указание.* Запись можно также выполнить функцией WriteFile().

* 1. Получить список ключей в хранилище. Формат команды:

list

Необходимо записать в канал строку, содержащую через пробел все имеющиеся в хранилище ключи.

* 1. Удалить значение под заданным ключом. Формат команды:

delete ключ

Если ключ присутствует в хранилище, следует записать в канал строку deleted, иначе — строку missing.

* 1. Прекратить сеанс связи. Формат команды:

quit

Необходимо отключить именованный канал от клиента функцией DisconnectNamedPipe().

1. Переходит к пункту 3 (цикл работы с подключенным клиентом).
2. Запрашивает у пользователя, следует ли остановить сервер. В случае утвердительного ответа следует уничтожить именованный канал, созданный на шаге 1, функцией CloseHandle().
3. Переходит к пункту 2 (цикл ожидания клиентов и работы с ними).

### Программа-клиент:

1. Запрашивает у пользователя имя канала и подключается как клиент к указанному каналу функцией CreateFile().
2. Запрашивает у пользователя строку-команду и записывает её в открытый канал функцией WriteFile().
3. Если на шаге 2 была введена команда quit, закрывает канал функцией CloseHandle() и завершает работу.
4. Считывает из канала сообщение-ответ функцией ReadFile() и отображает его на экране.
5. Переходит к шагу 2 (цикл).

*Указание 1.* Хранение значений по ключам в C++ удобно реализовать с std::map, а вычленение имени команды и аргументов из строки — классом std::stringstream.

*Указание 2.* При отладке первой из двух программ целесообразно использовать образец решения второй в качестве недостающей части. Раздел MSDN об именованных каналах также содержит [пример](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365592), близкий к программе-клиенту.

Вариант 3

Написать программу для обмена сообщениями через механизм Windows Mailstots, которая действует следующим образом:

1. Запрашивает у пользователя наименование почтового ящика и пытается создать его функцией CreateMailslot(), а если почтовый ящик уже существует, получает его описатель функцией CreateFile().

*Указание 1.* Пользователь должен вводить полное наименование почтового ящика, например, \\.\mailslot\test. Если оно не начинается с \\.\, такой ящик нельзя создать, к нему можно только подключиться (так как он расположен на удаленной машине).

*Указание 2.* Можно диагностировать, что почтовый ящик существует, если функция CreateMailslot() завершилась ошибкой с соответствующим кодом (который можно получить функцией GetLastError()). Узнать нужный код ошибки можно по MSDN.

1. Запрашивает у пользователя, какое действие следует выполнить:
   1. Получить информацию о почтовом ящике: количество сообщений, размер последнего сообщения (которое будет извлечено следующим), наибольший допустимый размер сообщения для данного ящика. Сведения необходимо вывести на экран, получив их функцией GetMailslotInfo().
   2. Поместить сообщение в почтовый ящик. Необходимо запросить у пользователя текст сообщения, который может быть многострочным и завершается пустой строкой, а затем записать сообщение в почтовый ящик функцией WriteFile().
   3. Получить сообщение из почтового ящика. Необходимо считать очередное сообщение функцией ReadFile() и отобразить его на экране.
   4. Завершить работу. Следует отключиться от почтового ящика, закрыв его описатель функцией CloseHandle().

*Указание.* По умолчанию созданный функцией CreateMailslot() почтовый ящик доступен для чтения и записи только ОС (учетной записи LOCAL SYSTEM). Параметром lpSecurityAttributes можно разрешить доступ всем процессам; код для заполнения структуры SECURITY\_ATRIBUTES приведен в [приложении](#_Приложение).

Вариант 4

Написать программу для обучения пользователя командной строки вежливости с использованием перенаправления потоков ввода и вывода через анонимные каналы.

1. Создать два анонимных канала функцией CreatePipe().
2. Запустить процесс cmd.exe, соединив один из созданных каналов с потоком ввода, а другой канал — с потоком вывода дочернего процесса.
3. Считать из канала, связанного с потоком вывода дочернего процесса, все данные функцией ReadFile() и вывести их на экран функцией WriteConsole().

*Указание 1.* Считать все данные можно, вызывая ReadFile() раз за разом и проверяя окончание считанных данных (их количество возвращается параметром lpNumberOfBytesRead). В условиях ЛР можно считать, что вывод окончен, если последний символ — '>'.

*Указание 2.* Первым аргументом функции WriteConsole() следует передать GetStdHandle(STD\_OUTPUT\_HANDLE).

1. Запросить у пользователя полную строку-команду.
2. Если введенная строка не начинается со слова «please» (до первого пробела), уведомить об этом пользователя и перейти к пункту 4.
3. Если введена строка «thanks», остановить дочерний процесс функцией TerminateProcess(), закрыть анонимные каналы функцией CloseHandle() и  завершить работу программы.
4. Записать в канал, связанный с потоком ввода дочернего процесса, оставшуюся часть команды и символ перевода строки '\n'.
5. Перейти к пункту 3 (цикл).

*Указание.* [Официальный пример](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682499) перенаправления стандартных потоков дочернего процесса через анонимные каналы может быть полезен в отношении использования отдельных функций, в частности, CreateFile().

Пример работы программы:

C:\> **date /T**

Please ask politely!

> **please date /T**

01.09.2015

C:\> **please do something nasty**

'do' is not recognized as an internal or external command,

operable program or batch file.

C:\> **thanks**

# Контрольные вопросы

1. В чем заключается механизм проецирования файлов в память и почему он может использоваться для межпроцессного взаимодействия?
2. Какими функциями Windows API и операциями языка программирования осуществляется использование файлов, проецируемых в память?
3. Что такое именованные каналы (named pipes) в Windows и почему они могут использоваться для межпроцессного взаимодействия?
4. Какие возможны в Windows варианты именованных каналов, каковы ограничения и преимущества каждого из них?
5. Какими функциями Windows API осуществляется использование именованных каналов в программе-сервере?
6. Какими функциями Windows API осуществляется использование именованных каналов в программе-клиенте?
7. Что такое анонимные каналы (anonymous pipes) и каким образом они могут использоваться для межпроцессного взаимодействия?
8. В чем заключается в Windows API наследование описателей (handle inheritance) и как это может использоваться для межпроцессного взаимодействия?
9. Какими функциями Windows API осуществляется работа с анонимными каналами и стандартными потоками ввода-вывода?
10. Что такое почтовые ящики (mailslots) в Windows API? Каковы их преимущества как средства межпроцессного взаимодействия?
11. Какими функциями Windows API осуществляется работа с mailslots?

# Приложение

Заполнение атрубутов доступа для разрешения чтения и записи всем пользователям, выполнившим вход в систему. После использования результата вызова функции create\_security\_attributes() можно функцией LocalFree() удалить результат create\_security\_descriptor().

/\* Необходимые заголовочные файлы. \*/

#define WINVER 0x0502

#include <windows.h>

#include <sddl.h>

static PSECURITY\_DESCRIPTOR create\_security\_descriptor()

{

const char\* sddl =

"D:(A;OICI;GRGW;;;AU)(A;OICI;GA;;;BA)";

PSECURITY\_DESCRIPTOR security\_descriptor = NULL;

ConvertStringSecurityDescriptorToSecurityDescriptor(

sddl, SDDL\_REVISION\_1, &security\_descriptor, NULL);

return security\_descriptor;

}

static SECURITY\_ATTRIBUTES create\_security\_attributes()

{

SECURITY\_ATTRIBUTES attributes;

attributes.nLength = sizeof(attributes);

attributes.lpSecurityDescriptor =

create\_security\_descriptor();

attributes.bInheritHandle = FALSE;

return attributes;

}