Белорусский государственный университет
Факультет прикладной математики и информатики
Кафедра технологии программирования
доцент Побегайло А.П.

5. Работа с анонимными каналами.

Так как работа с анонимными каналами требует совместного использования целого ряда функций, то данная глава организована следующим образом. Сначала рассмотрены все функции, которые предназначены для работы с анонимными каналами. После этого приведены примеры, иллюстрирующих использование этих функций. В последнем параграфе показано, как при помощи анонимных каналов можно перенаправить стандартный ввод-вывод. Изложение главы совпадает с порядком, который был предложен для работы с каналами в конце предыдущей главы.

5.1. Создание анонимных каналов.

Анонимные каналы создаются процессом сервером при помощи функции *CreatePipe*, которая имеет следующий прототип:

```
BOOL  CreatePipe (
        PHANDLE                 hReadHandle,            // дескриптор для чтения из канала
        PHANDLE                 hWriteHandle,           // дескриптор для записи в канал
        LPSECURITY_ATTRIBUTES   lpPipeAttributes,       // атрибуты защиты
        DWORD                   dwSize                  // размер буфера в байтах
);
```

При удачном завершении функция *CreatePipe* возвращает значение TRUE, а в случае неудачи – FALSE. Рассмотрим кратко назначение параметров этой функции. В случае успешного завершения функция *CreatePipe* возвращает в переменных, на которые указывают параметры hReadPipe и hWritePipe, два дескриптора. Дескриптор, на который указывает hReadPipe, в дальнейшем используется в функциях чтения данных из канала, а дескриптор hWritePipe – в функциях записи данных в канал. Параметр lpPipeAttributes определяет атрибуты защиты анонимного канала. Установку значения этого параметра lpPipeAttributes мы расссмотрим в следующем параграфе. Параметр dwSize определяет размер буфера ввода-вывода анонимного канала. Отметим, что операционные системы Windows автоматически определяют размер буфера и поэтому значение параметра dwSize является только пожеланием операционной системе при выборе размера буфера. Значение этого параметра можно установить равным 0, тогда операционная система выберет размер буфера по умолчанию.

5.2. Соединение клиентов с анонимным каналом.

Так как анонимные каналы не имеют имени, то для соединения процесса-клиента с таким каналом необходимо передать клиенту один из дескрипторов анонимного канала. При этом передаваемый дескриптор должен быть наследуемым. Наследование дескрипторов анонимного канала определяется значением поля bInheritHandle в структуре типа SECURITY_ATTRIBUTES, на которую указывает параметр lpPipeAttributes функции *CreatePipe*. Если значение этого поля, которое имеет тип BOOL, равно TRUE, то дескрипторы анонимного канала создаются наследуемыми, в противном случае – дескрипторы создаются ненаследуемыми. Если мы создаем наследуемые дескрипторы анонимного канала, то тот дескриптор, который не передается клиенту, должен быть сделан ненаследуемым. И наоборот, если мы создаем ненаследуемые дескрипторы анонимного канала, то тот дескриптор, который передается клиенту, должен быть сделан наследуемым. В операционной системе Windows 98 обе эти задачи решаются путем создания соответственно ненаследуемого или наследуемого дубликата исходного дескриптора, используя функцию *DuplicateHandle*. После этого исходный дескриптор закрывается. В операционной системе Windows 2000 эта задача может быть также решена, используя функцию *SetHandleInformation*, которая изменяет свойство наследования дескриптора.

Передача наследуемого дескриптора клиенту может выполняться одним из следующих способов:

– через командную строку;

– через поля hStdInput, hStdOutput и hStdError структуры STARTUPINFO;

– посредством сообщения WM_COPYDATA;
– через файл.

В данной главе мы будем использовать только первых два способа передачи дескрипторов процессу-клиенту. Третьим способом можно пользоваться только процессам с графическим интерфейсом (GUI). Поэтому этот способ передачи данных между процессами посредством сообщений будет рассмотрен в соответствующей главе. Передавать дескрипторы через файл мы не будем, так как предполагается, что через файлы передаются большие объемы общих данных. Более подробно вопросы наследования и дублирования дескрипторов были рассмотрены в главе 3.

5.3. Обмен данными по анонимному каналу.

Для обмена данными по анонимному каналу в операционных системах Windows используются те же функции, что для записи и чтения данных в файл. Для записи данных в анонимный канал используется функция *WriteFile*, которая имеет следующий прототип:

```
BOOL  WriteFile (
        HANDLE          hAnonymousPipe,          // дескриптор анонимного канала
        LPCVOID         lpBuffer,                // буфер данных
        DWORD           dwNumberOfBytesToWrite,  // число байт для записи
        LPDWORD         lpNumberOfBytesWritten,  // число записпнных байт
        LPOVERLAPPED    lpOverlapped             // асинхронный ввод
);
```

Функция *WriteFile* записывает в анонимный канал количество байт, заданных параметром dwNumberOfBytesToWrite, из буфера данных, на который указывает параметр lpBuffer. Дескриптор вывода этого анонимного канала должен быть задан первым параметром функции *WriteFile*. При успешном завершении функция *WriteFile* возвращает значение TRUE, а в случае неудачи – FALSE. Количество байт, записанных этой функцией в анонимный канал, возвращается в переменной, на которую указывает параметр lpNumberOfBytesWritten. Параметр lpOverlapped предназначен для выполнения асинхронной операции вывода, так как анонимные каналы поддерживают только синхронную передачу данных, то в нашем случае этот парамаетр всегда будет равен NULL.

Для чтения данных из анонимного канала используется функция *ReadFile*, которая имеет следующий прототип:

```
BOOL  ReadFile (
        HANDLE          hAnonymousPipe,          // дескриптор анонимного канала
        LPCVOID         lpBuffer,                // буфер данных
        DWORD           dwNumberOfBytesToRead,   // число байт для записи
        LPDWORD         lpNumberOfBytesRead,     // число записпнных байт
        LPOVERLAPPED    lpOverlapped             // асинхронный ввод
);
```

Функция *ReadFile* читает из анонимного канала количество байт, заданных параметром dwNumberOfBytesToRead, в буфер данных, на который указывает параметр lpBuffer. Дескриптор ввода этого ананимного канала должен быть задан первым параметром функции *ReadFile*. При успешном завершении функция *ReadFile* возвращает значение TRUE, а в случае неудачи – FALSE. Количество байт, прочитанных функцией *WriteFile* из анонимного канала, возвращается в переменной, на которую указывает параметр lpNumberOfBytesRead. Также как и в случае записи в анонимный канал параметр lpOverlapped должен быть равен NULL.

Отметим, что обмен данными по анонимному каналу осуществляется только в соответствии с назначением дескриптора этого канала. Дескриптор для записи в анонимный канал должен быть параметром функции *WriteFile*, а дескриптор для чтения из анонимного канала должен быть параметром функции *ReadFile*. В этом и состоит смысл передачи данных по анонимному каналу только в одном направлении. Однако это не означает, что один процесс может использовать анонимный канал только для записи или только для чтения. Один и тот же процесс может, как писать данные в анонимный канал, так и читать данные из него, должным образом используя дескрипторы этого анонимного канала.

В завершении этого параграфа отметим, что после завершения обмена данными по анонимному каналу, потоки должны закрыть дескрипторы записи и чтения анонимного канала, используя функцию *CloseHandle*.

5.5. Примеры работы с анонимными каналами.

Вначале рассмотрим простой пример, в котором процесс-сервер выполняет следующие действия:

- создает анонимный канал;
- создает дочерний процесс;
- передает созданному дочернему процессу один из дескрипторов созданного анонимного канала, используя для этого командную строку.

В этом случае дочерний процесс будет клиентом анонимного канала. Для определенности передадим клиенту дескриптор для записи в анонимный канал и оставим серверу дескриптор для чтения. Сначала приведем программу процесса-клиента анонимного канала.

```
// Пример процесса клиента анонимного канала.
// Клиент пишет в анонимный канал.
// Дескриптор анонимного канала передается клиенту через командную строку.

#include <windows.h>
#include <conio.h>

int main(int argc, char *argv[])
{
	HANDLE hWritePipe;

		// преобразуем символьное представление дескриптора в число
	hWritePipe = (HANDLE)atoi(argv[1]);
		// ждем команды о начале записи в анонимный канал
	_cputs("Press any key to start communication.\n");
	_getch();
		// пишем в анонимный канал
	for (int i = 0; i < 10; i++)
	{
		DWORD dwBytesWritten;
		if (!WriteFile(
				hWritePipe,
				&i,
				sizeof(i),
				&dwBytesWritten,
				NULL))
			{
				_cputs("Write to file failed.\n");
				_cputs("Press any key to finish.\n");
				_getch();
				return GetLastError();
			}
		_cprintf("The number %d is written to the pipe.\n", i);
		Sleep(500);
	}
		// закрываем дескриптор канала
	CloseHandle(hWritePipe);

	_cputs("The process finished writing to the pipe.\n");
	_cputs("Press any key to exit.\n");
	_getch();

	return 0;
}
```

Программа 5.1.

Теперь приведем программу процесса-сервера анонимного канала, который запускает клиента и передает ему через командную строку дескриптор записи в анонимный канал.

```
// Пример процесса сервера анонимного канала.
// Сервер читает из анонимного канала.
// Дескриптор анонимного канала передается
// клинету через командную строку.

#include <windows.h>
#include <conio.h>

int main()
{
	char lpszComLine[80];	// для командной строки

	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	HANDLE hWritePipe, hReadPipe, hInheritWritePipe;

		// создаем анонимный канал
	if(!CreatePipe(
			&hReadPipe,	// дескриптор для чтения
			&hWritePipe,	// дескриптор для записи
			NULL,		// атрибуты защиты по умолчанию,
					// в этом случае дескрипторы
					// hReadPipe и hWritePipe ненаследуемые
			0))		// размер буфера по умолчанию

	{
		_cputs("Create pipe failed.\n");
		_cputs("Press any key to finish.\n");
		_getch();
		return GetLastError();
	}
		// делаем наследуемый дубликат дескриптора hWritePipe
	if(!DuplicateHandle(
			GetCurrentProcess(),	// дескриптор текущего процесса
			hWritePipe,		// исходный дескриптор канала
			GetCurrentProcess(),	// дескриптор текущего процесса
			&hInheritWritePipe,	// новый дескриптор канала
			0,			// этот параметр игнорируется
			TRUE,			// новый декскриптор наследуемый
			DUPLICATE_SAME_ACCESS ))// доступ не изменяем
	{
		_cputs("Duplicate handle failed.\n");
		_cputs("Press any key to finish.\n");
		_getch();
		return GetLastError();
	}
		// закрываем ненужный дескриптор
	CloseHandle(hWritePipe);
		// устанавливаем атрибуты нового процесса
	ZeroMemory(&si, sizeof(STARTUPINFO));
	si.cb = sizeof(STARTUPINFO);
		// формируем командную строку
```

```
	wsprintf(lpszComLine, "C:\\Client.exe %d", (int)hInheritWritePipe);
		// запускаем новый консольный процесс
	if (!CreateProcess(
			NULL,		// имя процесса
			lpszComLine,	// командная строка
			NULL,		// атрибуты защиты процесса по умолчанию
			NULL,		// атрибуты защиты первичного потока по умолчанию
			TRUE,		// наследуемые дескрипторы текущего процесса
					// наследуются новым процессом
			CREATE_NEW_CONSOLE,	// новая консоль
			NULL,		// используем среду окружения процесса предка
			NULL,		// текущий диск и каталог, как и в процессе предке
			&si,		// вид главного окна - по умолчанию
			&pi		// здесь будут дескрипторы и идентификаторы
					// нового процесса и его первичного потока
			))
		{
			_cputs("Create process failed.\n");
			_cputs("Press any key to finish.\n");
			_getch();
			return GetLastError();
		}
		// закрываем дескрипторы нового процесса
	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);
		// закрываем ненужный дескриптор канала
	CloseHandle(hInheritWritePipe);
		// читаем из анонимного канала
	for (int i = 0; i < 10; i++)
	{
		int nData;
		DWORD dwBytesRead;
		if (!ReadFile(
				hReadPipe,
				&nData,
				sizeof(nData),
				&dwBytesRead,
				NULL))
			{
				_cputs("Read from the pipe failed.\n");
				_cputs("Press any key to finish.\n");
				_getch();
				return GetLastError();
			}
		_cprintf("The number %d is read from the pipe.\n", nData);
	}
		// закрываем дескриптор канала
	CloseHandle(hReadPipe);

	_cputs("The process finished reading from the pipe.\n");
	_cputs("Press any key to exit.\n");
	_getch();

	return 0;
}
```

Программа 5.2.

В следующих программах показывается, как организовать двусторонний обмен данными по анонимному каналу между клиентом и сервером. Для этого дескрипторы чтения и записи в анонимный канал

используются как сервером, так и клиентом этого анонимного канала. В этом примере также сначала приведена программа процесса-клиента анонимного канала.

```
// Пример процесса клиента анонимного канала.
// Клиент сначала пишет в анонимный канал, а потом читает из него.
// Дескриптор анонимного канала передается клиенту через командную строку.

#include <windows.h>
#include <conio.h>

int main(int argc, char *argv[])
{
	HANDLE hWritePipe, hReadPipe;
	HANDLE hEnableRead;		// для синхронизации обмена данными
	char lpszEnableRead[] = "EnableRead";

		// открываем событие, разрешающее чтение
	hEnableRead = OpenEvent(EVENT_ALL_ACCESS, FALSE, lpszEnableRead);

		// преобразуем символьное представление дескрипторов в число
	hWritePipe = (HANDLE)atoi(argv[1]);
	hReadPipe = (HANDLE)atoi(argv[2]);
		// ждем команды о начале записи в анонимный канал
	_cputs("Press any key to start communication.\n");
	_getch();
		// пишем в анонимный канал
	for (int i = 0; i < 10; i++)
	{
		DWORD dwBytesWritten;
		if (!WriteFile(
				hWritePipe,
				&i,
				sizeof(i),
				&dwBytesWritten,
				NULL))
			{
				_cputs("Write to file failed.\n");
				_cputs("Press any key to finish.\n");
				_getch();
				return GetLastError();
			}
		_cprintf("The number %d is written to the pipe.\n", i);
	}
	_cputs("The process finished writing to the pipe.\n");

		// ждем разрешения на чтение
	WaitForSingleObject(hEnableRead, INFINITE);
		// читаем ответ из анонимного канала
	for (int j = 0; j < 10; j++)
	{
		int nData;
		DWORD dwBytesRead;
		if (!ReadFile(
				hReadPipe,
				&nData,
				sizeof(nData),
				&dwBytesRead,
				NULL))
			{
```

```
                        _cputs("Read from the pipe failed.\n");
                        _cputs("Press any key to finish.\n");
                        _getch();
                        return GetLastError();
                    }
                _cprintf("The number %d is read from the pipe.\n", nData);
        }
        _cputs("The process finished reading from the pipe.\n");
        _cputs("Press any key to exit.\n");
        _getch();

                // закрываем дескрипторы канала
        CloseHandle(hWritePipe);
        CloseHandle(hReadPipe);
        CloseHandle(hEnableRead);

        return 0;
}
```

Программа 5.3.

Теперь приведем текст программы процесса-сервера анонимного канала, который запускает клиента и передает ему дескрипторы анонимного канала через командную строку.

```
// Пример процесса сервера анонимного канала.
// Сервер сначала читает из анонимного канала, а затем пишет в него.
// Дескриптор анонимного канала передается клиенту через командную строку.

#include <windows.h>
#include <conio.h>

int main()
{
        char lpszComLine[80];     // для командной строки

        HANDLE hEnableRead;             // для синхронизации обмена данными
        char lpszEnableRead[] = "EnableRead";

        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        HANDLE hWritePipe, hReadPipe;
        SECURITY_ATTRIBUTES sa;

                // создаем событие для синхронизации обмена данными
        hEnableRead = CreateEvent(NULL, FALSE, FALSE, lpszEnableRead);

                // устанавливает атрибуты защиты канала
        sa.nLength = sizeof(SECURITY_ATTRIBUTES);
        sa.lpSecurityDescriptor = NULL;             // защита по умолчанию
        sa.bInheritHandle = TRUE;                   // дескрипторы наследуемые

                // создаем анонимный канал
        if(!CreatePipe(
                        &hReadPipe,     // дескриптор для чтения
                        &hWritePipe,    // дескриптор для записи
                        &sa,            // атрибуты защиты по умолчанию,
                                        // дескрипторы наследуемые
                        0))             // размер буфера по умолчанию
```

```
	{
		_cputs("Create pipe failed.\n");
		_cputs("Press any key to finish.\n");
		_getch();
		return GetLastError();
	}
	// устанавливаем атрибуты нового процесса
ZeroMemory(&si, sizeof(STARTUPINFO));
si.cb = sizeof(STARTUPINFO);
	// формируем командеую строку
wsprintf(lpszComLine, "C:\\Client.exe %d %d",
				(int)hWritePipe, (int)hReadPipe);
	// запускаем новый консольный процесс
if (!CreateProcess(
		NULL,		// имя процесса
		lpszComLine,	// командная строка
		NULL,		// атрибуты защиты процесса по умолчанию
		NULL,		// атрибуты защиты первичного потока по умолчанию
		TRUE,		// наследуемые дескрипторы текущего процесса
				// наследуются новым процессом
		CREATE_NEW_CONSOLE,	// новая консоль
		NULL,		// используем среду окружения процесса предка
		NULL,		// текущий диск и каталог, как и в процессе предке
		&si,		// вид главного окна - по умолчанию
		&pi		// здесь будут дескрипторы и идентификаторы
				// нового процесса и его первичного потока
		))
	{
		_cputs("Create process failed.\n");
		_cputs("Press any key to finish.\n");
		_getch();
		return GetLastError();
	}
	// закрываем дескрипторы нового процесса
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
	// читаем из анонимного канала
for (int i = 0; i < 10; i++)
{
	int nData;
	DWORD dwBytesRead;
	if (!ReadFile(
			hReadPipe,
			&nData,
			sizeof(nData),
			&dwBytesRead,
			NULL))
		{
			_cputs("Read from the pipe failed.\n");
			_cputs("Press any key to finish.\n");
			_getch();
			return GetLastError();
		}
	_cprintf("The number %d is read from the pipe.\n", nData);
}
_cputs("The process finished reading from the pipe.\n");

	// даем сигнал на разрешение чтения клиентом
SetEvent(hEnableRead);
```

```
		// пишем ответ в анонимный канал
	for (int j = 10; j < 20; j++)
	{
		DWORD dwBytesWritten;
		if (!WriteFile(
				hWritePipe,
				&j,
				sizeof(j),
				&dwBytesWritten,
				NULL))
			{
				_cputs("Write to file failed.\n");
				_cputs("Press any key to finish.\n");
				_getch();
				return GetLastError();
			}
		_cprintf("The number %d is written to the pipe.\n", j);
	}
		// закрываем дескрипторы канала
	CloseHandle(hReadPipe);
	CloseHandle(hWritePipe);
	CloseHandle(hEnableRead);

	_cputs("The process finished writing to the pipe.\n");
	_cputs("Press any key to exit.\n");
	_getch();

	return 0;
}
```

Программа 5.4.

Отметим в последнем примере следующий момент. Для организации двустороннего обмена данными по анонимному каналу, сервер и клиенты анонимного канала должны синхронизировать доступ к этому каналу. То есть для организации передачи данных необходимо разработать протокол передачи данных или использовать объекты синхронизации, которые исключают одновременный неконтролируемый доступ параллельных потоков к анонимному каналу. В приведенном примере событие hEnableRead сигнализирует клиенту, что сервер закончил чтение данных и теперь данные из канала может читать клиент. При отсутствии такой синхронизации возможно одновременное чтение данных сервером и клиентом, так как они работают параллельно, что вызовет неправильную работу программы и её зависание.

5.5. Перенаправление стандартного ввода-вывода.

Анонимные каналы часто используются для перенаправления стандартного ввода-вывода. Чтобы подробнее разобраться с этим вопросом, сначала кратко рассмотрим стандартные средства ввода-вывода, используемые в языке C++. Компилятор языка C++ фирмы Microsoft содержит стандартную библиотеку, которая поддерживает три варианта функций стандартного ввода-вывода. Описания этих функций находятся в следующих заголовочных файлах: <stdio.h>, <iostream.h> и <conio.h>. Функции ввода-вывода, которые описаны в заголовке <stdio.h>, обеспечивают ввод-вывод в следующие стандартные потоки:

stdin – стандартный файл ввода;
stdout – стандартный файл вывода;
stderr – файл вывода сообщений об ошибках.

Эти функции составляют стандартную библиотеку ввода-вывода языка C. Функции и операторы ввода-вывода, которые описаны в заголовке <iostream.h>, обеспечивают ввод-вывод в стандартные потоки ввода-вывода cin, cout, cerr. Эти функции составляют стандартную библиотеку ввода-вывода языка C++. При создании консольного процесса или при распределении консоли приложением с графическим интерфейсом, стандартные потоки ввода-вывода связываются с дескрипторами, которые заданы в полях hStdInput, hStdOutput и hStdError структуры типа STARTUPINFO. Поэтому, если в эти поля будут записаны соответствующие дескрипторы

анонимного канала, то для передачи данных по анонимному каналу можно использовать функции стандартного ввода-вывода. Такая процедура называется *перенаправлением стандартного ввода-вывода*.

Функции ввода-вывода, которые поддерживаются заголовком <conio.h>, отличаются от функций стандартной библиотеки ввода-вывода языка С только тем, что они всегда связываются с консолью. Поэтому эти функции можно использовать для ввода-вывода на консоль даже в случае перенаправления стандартного ввода-вывода.

Ниже приведены программы, в которых стандартный ввод-вывод перенаправляется в анонимный канал, а для обмена данными по анонимному каналу используются перегруженные операторы ввода-вывода. Пример включает программы следующих процессов: два процесса клиента, которые обмениваются данными по анонимному каналу, и процесс сервер, который создает клиентов и передает им дескрипторы анонимного канала через поля структуры STARTUPINFO. Сначала приведем программы, которые описывают процессы клиенты.

```
// Пример обмена данными по анонимному каналу,
// используя перенаправленные стандартные потоки ввода-вывода.
// Дескрипторы анонимного канала передаются через поля структуры STARTUPINFO.

#include <windows.h>
#include <conio.h>
#include <iostream.h>

int main()
{
		// события для синхронизации обмена данными
	HANDLE hReadFloat, hReadText;
	char lpszReadFloat[] = "ReadFloat";
	char lpszReadText[] = "ReadText";

		// открываем события
	hReadFloat = CreateEvent(NULL, FALSE, FALSE, lpszReadFloat);
	hReadText = CreateEvent(NULL, FALSE, FALSE, lpszReadText);

		// ждем команды о начале записи в анонимный канал
	_cputs("Press any key to start communication.\n");
	_getch();
		// пишем целые числа в анонимный канал
  for (int i = 0; i < 5; ++i)
	{
		Sleep(500);
		cout << i << endl;
	}

		// ждем разрешение на чтение дробных чисел из канала
	WaitForSingleObject(hReadFloat, INFINITE);
		// читаем дробные числа из анонимного канала
  for (int j = 0; j < 5; ++j)
	{
		float nData;
		cin >> nData;
		_cprintf("The number %2.1f is read from the pipe.\n", nData);
  }

		// отмечаем, что можно читать текст из анонимного канала
	SetEvent(hReadText);
		// теперь передаем текст
	cout << "This is a demo sentence." << endl;
		// отмечаем конец передачи
	cout << '\0' << endl;
```

```
	_cputs("The process finished transmission of data.\n");
	_cputs("Press any key to exit.\n");
	_getch();

	CloseHandle(hReadFloat);
	CloseHandle(hReadText);

	return 0;
}
```

Программа 5.5.

```
// Пример обмена данными по анонимному каналу.
// используя перенаправленные стандартные потоки ввода-вывода.
// Дескрипторы анонимного канала передаются через поля структуры STARTUPINFO.

#include <windows.h>
#include <conio.h>
#include <iostream.h>

int main()
{
		// события для синхронизации обмена данными
	HANDLE hReadFloat, hReadText;
	char lpszReadFloat[] = "ReadFloat";
	char lpszReadText[] = "ReadText";

		// открываем события
	hReadFloat = CreateEvent(NULL, FALSE, FALSE, lpszReadFloat);
	hReadText = CreateEvent(NULL, FALSE, FALSE, lpszReadText);

		// читаем целые числа из анонимного канала
	for (int i = 0; i < 5; ++i)
	{
		int nData;
		cin >> nData;
		_cprintf("The number %d is read from the pipe.\n", nData);
	}

		// разрешаем читать дробные числа из анонимного канала
	SetEvent(hReadFloat);
		// пишем дробные числа в анонимный канал
	for (int j = 0; j < 5; ++j)
	{
		Sleep(500);
		cout << (j*0.1) << endl;
	}

		// ждем разрешения на чтение текста
	WaitForSingleObject(hReadText, INFINITE);
	_cputs("The process read the text: ");
		// теперь читаем текст
	char lpszInput[80];
	do
	{
		Sleep(500);
		cin >> lpszInput;
		_cputs(lpszInput);
```

```
            _cputs(" ");
        }
        while (*lpszInput != '\0');

        _cputs("\nThe process finished transmission of data.\n");
        _cputs("Press any key to exit.\n");
        _getch();

        CloseHandle(hReadFloat);
        CloseHandle(hReadText);

        return 0;
}
```

Программа 5.6.

Теперь приведем программу, которая описывает сервер анонимного канала. Эта программа просто создает двух клиентов анонимного канала и прекращает свою работу.

```
// Пример процесса сервера анонимного канала.
// Сервер создает анонимный канал, а затем два процесса клиента
// анонимного канала, которые обмениваются между собой данными по этому каналу.
// Дескрипторы анонимного канала передаются клиентам через поля структуры STARTUPINFO.

#include <windows.h>
#include <conio.h>

int main()
{
        char lpszComLine1[80] = "C:\\Client1.exe"; // имя первого клиента
        char lpszComLine2[80] = "C:\\Client2.exe"; // имя второго клиента

        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        HANDLE hWritePipe, hReadPipe;
        SECURITY_ATTRIBUTES sa;

                // устанавливает атрибуты защиты канала
        sa.nLength = sizeof(SECURITY_ATTRIBUTES);
        sa.lpSecurityDescriptor = NULL;    // защита по умолчанию
        sa.bInheritHandle = TRUE;                    // дескрипторы наследуемые

                // создаем анонимный канал
        if(!CreatePipe(
                        &hReadPipe,      // дескриптор для чтения
                        &hWritePipe,     // дескриптор для записи
                        &sa,             // атрибуты защиты по умолчанию,
                                         // дескрипторы наследуемые
                        0))              // размер буфера по умолчанию

        {
                _cputs("Create pipe failed.\n");
                _cputs("Press any key to finish.\n");
                _getch();
                return GetLastError();
        }
                // устанавливаем атрибуты нового процесса
        ZeroMemory(&si, sizeof(STARTUPINFO));
```

```
si.cb = sizeof(STARTUPINFO);
        // использовать стандартные дескрипторы
si.dwFlags = STARTF_USESTDHANDLES;
        // устанавливаем стандартные дескрипторы
si.hStdInput = hReadPipe;
si.hStdOutput = hWritePipe;
si.hStdError = hWritePipe;
        // запускаем первого клиента
if (!CreateProcess(
                NULL,           // имя процесса
                lpszComLine1,   // командная строка
                NULL,           // атрибуты защиты процесса по умолчанию
                NULL,           // атрибуты защиты первичного потока по умолчанию
                TRUE,           // наследуемые дескрипторы текущего процесса
                                // наследуются новым процессом
                CREATE_NEW_CONSOLE,     // создаем новую консоль
                NULL,           // используем среду окружения процесса предка
                NULL,           // текущий диск и каталог как и в процессе предке
                &si,            // вид главного окна - по умолчанию
                &pi             // здесь будут дескрипторы и идентификаторы
                                // нового процесса и его первичного потока
                )
        )
{
        _cputs("Create process failed.\n");
        _cputs("Press any key to finish.\n");
        _getch();
        return GetLastError();
}
        // закрываем дескрипторы первого клиента
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);

        // запускаем второго клиента
if (!CreateProcess(
                NULL,           // имя процесса
                lpszComLine2,   // командная строка
                NULL,           // атрибуты защиты процесса по умолчанию
                NULL,           // атрибуты защиты первичного потока по умолчанию
                TRUE,           // наследуемые дескрипторы текущего процесса
                                // наследуются новым процессом
                CREATE_NEW_CONSOLE,     // создаем новую консоль
                NULL,           // используем среду окружения процесса предка
                NULL,           // текущий диск и каталог как и в процессе предке
                &si,            // вид главного окна - по умолчанию
                &pi             // здесь будут дескрипторы и идентификаторы
                                // нового процесса и его первичного потока
                )
        )
{
        _cputs("Create process failed.\n");
        _cputs("Press any key to finish.\n");
        _getch();
        return GetLastError();
}
        // закрываем дескрипторы второго клиента
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);

        // закрываем дескрипторы канала
```

```
        CloseHandle(hReadPipe);
        CloseHandle(hWritePipe);

    _cputs("The clients are created.\n");
        _cputs("Press any key to exit.\n");
        _getch();

        return 0;
}
```

Программа 5.7.