ВГКС
Кафедра ПОСТ
Курс «Системное программное обеспечение»
Лабораторная работа №6 (4 часа)

Тема: «Обмен данными по именованному каналу с сервером».

6. Работа с именованными каналами.

Работа с именованными каналами также как и работа с анонимными каналами требует совместного использования целого ряда функций. Поэтому сначала рассмотрены все функции, которые предназначены для работы с именованными каналами, а затем приведены несколько примеров, которые иллюстрируют использование этих функций.

6.1. Создание именованных каналов.

Именованные каналы создаются процессом-сервером при помощи функции *CreateNamedPipe*, которая имеет следующий прототип:

```
HANDLE  CreateNamedPipe (
        LPCTSTR         lpName,                 // имя канала
        DWORD           dwOpenMode,             // атрибуты канала
        DWORD           dwPipeMode,             // режим передачи данных
        DWORD           nMaxInstances,          // максимальное количество экземпляров канала
        DWORD           nOutBufferSize,         // размер выходного буфера
        DWORD           nInBufferSize,          // размер входного буфера
        DWORD           nDefaultTimeOut,        // время ожидания связи с клиентом
        LPSECURITY_ATTRIBUTES   lpPipeAttributes        // атрибуты защиты
);
```

где параметры имеют следующие значения.

Параметр lpName указывает на строку, которая должна иметь вид:

\\.\pipe\<pipe_name>

Здесь точка (.) обозначает локальную машину, так как новый именованный канал всегда создается на локальной машине, слово pipe – фиксировано, а <pipe_name> обозначает имя канала, которое задается пользователем и нечувствительно к регистру.

Параметр dwOpenMode задает флаги, которые определяют направление передачи данных, буферизацию, синхронизацию обмена данными и права доступа к именованному каналу. Для определения направления передачи данных используются флаги:

PIPE_ACCESS_DUPLEX	чтение и запись в канал,
PIPE_ACCESS_INBOUND	клиент пишет, а сервер читает данные,
PIPE_ACCESS_OUTBOUND	сервер пишет, а клиент читает данные.

Флаг, определяющий направление передачи данных по именованному каналу, должен совпадать для всех экземпляров одного и того же именованного канала. Для определения способа буферизации и синхронизации используются флаги:

FILE_FLAG_WRITE_THROUGH	запрещает буферизацию при передаче данных по сети.
FILE_FLAG_OVERLAPPED	разрешает асинхронную передачу данных по каналу.

Эти флаги могут быть разными для каждого экземпляра одного и того же именованного канала. Флаги для определения атрибутов защиты будут рассмотрены позднее.

Параметр dwPipeMode задает флаги, способ передачи данных по именованному каналу. Для определения способов чтения и записи данных в именованный канал используются флаги:

PIPE_TYPE_BYTE	запись данных потоком,
PIPE_TYPE_MESSAGE	запись данных сообщениями.
PYPE_READMODE_BYTE	чтение данных потоком,
PYPE_READMODE_MESSAGE	чтение данных сообщениями.

По умолчанию данные по именованному каналу предаются потоком. Флаги способов чтения и записи данных в именованный канал должны совпадать для всех экземпляров одного и того же именованного канала. Для определения синхронизации доступа к именованному каналу используются флаги:

PIPE_WAIT	синхронная связь с каналом и обмен данными по каналу,
PIPE_NOWAIT	асинхронная связь с каналом и обмен данными по каналу.

Эти флаги могут быть разными для каждого экземпляра именованного канала.

Параметр nMaxInstances определяет максимальное число экземпляров именованного канала, которое может находиться в пределах от 1 до PIPE_UNLIMITED_INSTANCES.

Параметры nOutBufferSize и nInBufferSize определяют соответственно размеры выходного и входного буферов для обмена данными по именованному каналу. Однако, эти значения рассматриваются операционными системами Windows только как пожелания пользователя, а сам выбор размеров буферов остается за операционной системой.

Параметр nDefaultTimeOut устанавливает время ожидания клиентом связи с сервером, если клиент вызывает функцию *WaitNamedPipe*, в которой интервал ожидания интервал ожидания задается по умолчанию.

При удачном завершение функция *CreateNamedPipe* возвращает значение дескриптор именованного канала, в случае неудачи – одно из двух значений:

INVALID_HANDLE_VALUE	неудачное завершение,
ERROR_INVALID_PARAMETR	значение параметра nMaxInstances больше, чем величина PIPE_UNLIMITED_INSTANCES.

Для связи сервера с несколькими клиентами по одному именованному каналу сервер должен создать несколько экземпляров этого канала. Каждый экземпляр именованного канала создается вызовом функции *CreateNamedPipe*, в которой некоторые флаги должны быть установлены одинаково для всех экземпляров одного и того же именованного канала. Каждый новый вызов этой функции возвращает новый дескриптор на создаваемый экземпляр именованного канала.

6.2. Соединение сервера с клиентом.

После того, как сервер создал именованный канал, он должен дождаться соединения клиента с этим каналом. Для этого сервер вызывает функцию

```
BOOL  ConnectNamedPipe (
        HANDLE          hNamedPipe,         // дескриптор канала
        LPOVERLAPPED    lpOverlapped        // асинхронная связь
);
```

которая возвращает значение TRUE в случае успеха или значение FALSE в случае неудачи. Сервер может использовать эту функцию для связи с клиентом по каждому новому экземпляру именованного канала.

После окончания обмена данными с клиентом, сервер может вызвать функцию

```
BOOL  DisconnectNamedPipe (
        HANDLE          hNamedPipe          // дескриптор канала
);
```

которая возвращает значение TRUE в случае успеха или значение FALSE в случае неудачи. Эта функция разрывает связь сервера с клиентом. После этого клиент не может обмениваться данными с сервером по данному именованному каналу и поэтому любая операция доступа к именованному каналу со стороны клиента вызовет ошибку. После разрыва связи с одним клиентом, сервер снова может вызвать функцию *ConnectNamedPipe*, чтобы установить связь по этому же именованному каналу с другим клиентом.

6.3. Соединение клиентов с именованным каналом.

Прежде чем соединяться с именованным каналом, клиент должен определить доступен ли какой-либо экземпляр этого канала для соединения. С этой целью клиент должен вызвать функцию:

```
BOOL  WaitNamedPipe (
        LPCTSTR         lpNamedPipeName,        // указатель на имя канала
        DWORD           nTimeOut                // интервал ожидания
);
```

которая в случае успешного завершения возвращает значение TRUE, а в случае неудачи – FALSE. Параметры этой функции имеют следующие значения.

Параметр lpNamedPipeName указывает на строку, которая должна иметь вид

\\<server_name>\pipe\<pipe_name>

Здесь <server_name> обозначает имя компьютера, на котором выполняется сервер именованного канала.

Параметр nTimeOut задает временной интервал в течение которого клиент ждет связь с сервером. Этот временной интервал определяется в миллисекундах или может быть равен одному из следующих значений:

NMPWAIT_USE_DEFAULT_WAIT интервал времени ожидания определяется значением параметра nDefaultTimeOut, который задается в функции *CreateNamedPipe*,

NMPWAIT_WAIT_FOREVER бесконечное время ожидания связи с именованным каналом.

Сделаем два важных замечания относительно работы функции *WaitNamedPipe*. Во-первых, если не существует экземпляров именованного канала с именем lpNamedPipe, то эта функция немедленно завершается неудачей, независимо от времени ожидания, заданного параметром nTimeOut. Во-вторых, если клиент соединяется с каналом до вызова сервером функции *ConnectNamedPipe*, то функция *WaitNamedPipe* возвращает значение FALSE и функция *GetLastError* вернет код ERROR_PIPE_CONNECTED. Поэтому функцию *WaitNamedPipe* нужно вызывать только после соединения сервера с каналом посредством функции *ConnectNamedPipe*.

После того как обнаружен свободный экземпляр канала, для того чтобы установить связь с этим каналом клиент должен вызвать функцию

```
HANDLE  CreateFile (
        LPCTSTR                 lpFileName,             // указатель на имя канала
        DWORD                   dwDesiredAccess,        // чтение или запись в канал
        DWORD                   dwShareMode,            // режим совместного использования
        LPSECURITY_ATTRIBUTES   lpSecurityAttributes,   // атрибуты защиты
        DWORD                   dwCreationDisposition,  // флаг открытия канала
        DWORD                   dwFlagsAndAttributes,   // флаги и атрибуты
        HANDLE                  hTemplateFile           // дополнительные атрибуты
);
```

которая в случае успешного завершения возвращает дескриптор именованного канала, а в случае неудачи – значение INVALID_HANDLE_VALUE.

Параметры функции *CreateFile* могут принимать следующие значения, если эта функция используется для открытия именованного канала.

Параметр lpFileName должен указывать на имя канала, которое должно быть задано в том же формате, что и в функции *WaitNamedPipe*.

Параметр dwDesiredAccess может принимать одно из следующих значений:

0	разрешает получить атрибуты канала,
GENERIC_READ	разрешает чтение из канала,
GENERIC_WRITE	разрешает запись в канал.

Следует отметить, что функция *CreateFile* завершается неудачей, если доступ к именованному каналу, заданный этими значениями, не соответствует значениям параметра dwOpenMode в функции *CreateNamedPipe*. Кроме того, в этом параметре программист может определить стандартные права доступа к именованному каналу. За более подробной информацией по этому вопросу нужно обратиться к MSDN.

Параметр dwShareMode определяет режим совместного использования именованного канала и может принимать значение 0, которое запрещает совместное использование именованного канала или любую комбинацию следующих значений:

FILE_SHARE_READ	разрешает совместное чтение из канала,
FILE_SHARE_WRITE	разрешает совместную запись в канал.

Параметр lpSecurityAttributes задает атрибуты защиты именованного канала.

Для именованного канала параметр dwCreationDisposition должен быть равен значению OPEN_EXISTING, так как клиент всегда открывает существующий именованный канал.

Для именованного канала параметр dwFlagsAndAttributes можно задается равным 0, что определяет флаги и атрибуты по умолчанию. Подробную информацию о значениях этого параметра смотри в MSDN.

Значение параметра hTemplateFile задается равным NULL.

Сделаем следующие замечания относительно работы с функцией *CreateFile* в случае её использования для открытия доступа к именованному каналу. Во-первых, несмотря на то, что функция *WaitNamedPipe* может успешно завершиться, последующий вызов функции *CreateFile* может завершиться неудачей по следующим причинам:

между вызовами этих функций сервер закрыл канал,

между вызовами функций другой клиент связался с экземпляром этого канала.

Для предотвращения последней ситуации сервер должен создавать новый экземпляр именованного канала после каждого успешного завершения функции *ConnectNamedPipe* или создать сразу несколько экземпляров именованного канала. Во-вторых, если заранее известно, что сервер вызвал функцию *ConnectNamedPipe*, то функция *CreateFile* может вызываться без предварительного вызова функции *WaitNamedPipe*.

Кроме того следует отметить, что если клиент работает на той же машине, что и сервер и использует для открытия именованного канала в функции *CreateFile* имя сервера в виде:

\\.\pipe\<pipe_name>

то файловая система именованных каналов (NPFS) открывает этот именованный канал в режиме передачи данных потоком. Чтобы открыть именованный канал в режиме передачи данных сообщениями, нужно задавать имя сервера в виде:

\\<server_name>\pipe\<pipe_name>

Отметим один момент, который касается связи сервера с клиентом именованного канала. Может возникнуть такая ситуация, что сервер вызвал функцию *ConnectNamedPipe*, а клиента, который хочет связаться с именованным каналом, не существует. В этом случае серверное приложение будет заблокировано. Чтобы иметь возможность обработать такую ситуацию, функцию *ConnectNamedPipe* следует вызывать в отдельном потоке серверного приложения. Тогда для разблокировки серверного приложения можно вызвать функцию для связи клиента с именованным каналом из другого потока.

6.4. Получение информации об именованном канале.

Для получения информации о режимах работы и состоянии именованного канала используются функции:

GetNamedPipeHandleState;
GetNamedPipeInfo;

6.5. Изменение состояния именованного канала.

Изменить состояние именованного канала можно посредством функции

SetNamedPipeHandleState;

6.6. Обмен данными по именованному каналу.

Как и в случае с анонимным каналом, для обмена данными по именованному каналу используются функции *ReadFile* и *WriteFile*, но с одним отличием, которое заключается в следующем. Так как в случае именованного канала разрешен асинхронный обмен данными, то в функциях *ReadFile* и *WriteFile* может использоваться параметр lpOverlapped при условии, что в вызове функции *CreateNamedPipe* в параметре dwOpenMode был установлен флаг FILE_FLAG_OVERLAPPED.

Для асинхронного ввода-вывода по именованному каналу могут также использоваться функции *ReadFileEx* и *WriteFileEx*, которые будут рассмотрены далее в одной из глав.

Для копирования данных из именованного канала используется функция *PeekNamedPipe*, которая копирует данные в буфер, не удаляя их из канала. Эта функция имеет следующий прототип:

PeekNamedPipe

Для обмена сообщениями по сети может также использоваться функция *TransactNamedPipe*, которая объединяет операции записи и чтения в одну операцию (транзакцию) и имеет следующий прототип:

TransactNamedPipe

Параметры этой функции аналогичны параметрам функций *ReadFile* и *WriteFile*. Отметим, что функция *TransactNamedPipe* может использоваться только в том случае, если сервер при создании именованного канала установил флаги PIPE_TYPE_MESSAGE и PIPE_READMODE_MESSAGE.

Для передачи единственной транзакции по именованному каналу используется функция *CallNamedPipe*.

После завершения обмена данными по именованному каналу, потоки должны закрыть дескрипторы экземпляров именованного канала, используя функцию *CloseHandle*.

6.7. Примеры работы с именованными каналами.

Вначале рассмотрим простой пример, в котором процесс-сервер создает именованный канал, а затем ждет, пока клиент не соединится с именованным каналом. После этого сервер читает из именованного канала десять чисел и выводит их на консоль. Сначала приведем программу процесса-сервера именованного канала.

```
// Пример процесса сервера именованного канала.

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

int main()
{
	char c;		// служебный символ
	HANDLE  hNamedPipe;

		// создаем именованный канал для чтения
	hNamedPipe=CreateNamedPipe(
		"\\\\.\\pipe\\demo_pipe",    // имя канала
		PIPE_ACCESS_INBOUND,			// читаем из канала
		PIPE_TYPE_MESSAGE | PIPE_WAIT,	// синхронная передача сообщений
		1,			// максимальное количество экземпляров канала
		0,			// размер выходного буфера по умолчанию
		0,			// размер входного буфера по умолчанию
		INFINITE,	// клиент ждет связь бесконечно долго
		(LPSECURITY_ATTRIBUTES)NULL	// защита по умолчанию
	);
		// проверяем на успешное создание
	if (hNamedPipe==INVALID_HANDLE_VALUE)
	{
		cerr << "Creation of the named pipe failed." << endl
			<< "The last error code: " << GetLastError() << endl;
		cout << "Press any char to finish server: ";
		cin >> c;
		return 0;
	}

		// ждем пока клиент свяжется с каналом
	cout << "The server is waiting for connection with a client." << endl;
	if(!ConnectNamedPipe(
			hNamedPipe,		// дескриптор канала
			(LPOVERLAPPED)NULL		// связь синхронная
		))
	{
		cerr << "The connection failed." << endl
			<< "The last error code: " << GetLastError() << endl;
```

```
		CloseHandle(hNamedPipe);
		cout << "Press any char to finish the server: ";
		cin >> c;
		return 0;
	}
		// читаем данные из канала
	for (int i=0; i<10; i++)
	{
		int nData;
		DWORD dwBytesRead;
		if (!ReadFile(
				hNamedPipe,		// дескриптор канала
				&nData,			// адрес буфера для ввода данных
				sizeof(nData),		// количество читаемых байтов
				&dwBytesRead,		// количество прочитанных байтов
				(LPOVERLAPPED)NULL	// передача данных синхронная
			))
		{
			cerr << "Data reading from the named pipe failed." << endl
				<< "The last error code: " << GetLastError() << endl;
			CloseHandle(hNamedPipe);
			cout << "Press any char to finish the server: ";
			cin >> c;
			return 0;
		}
			// выводим прочитанные данные на консоль
		cout << "The number " << nData << " was read by the server" << endl;
	}
		// закрываем дескриптор канала
	CloseHandle(hNamedPipe);
		// завершаем процесс
	cout << "The data are read by the server."<<endl;
	cout << "Press any char to finish the server: ";
	cin >> c;
	return 0;
}
```

Программа 6.1. Пример сервера именованного канала.

Теперь приведем пример клиента именованного канала, который сначала связывается с именованным каналом, а затем записывает в него десять чисел.

```
// Пример процесса клиента именованного канала.

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

int main()
{
	char  c;		// служебный символ
	HANDLE  hNamedPipe;
	char pipeName[] = "\\\\.\\pipe\\demo_pipe";

		// связываемся с именованным каналом
	hNamedPipe = CreateFile(
		pipeName,			// имя канала
		GENERIC_WRITE,			// записываем в канал
		FILE_SHARE_READ,		// разрешаем только запись в канал
		(LPSECURITY_ATTRIBUTES) NULL,	// защита по умолчанию
		OPEN_EXISTING,			// открываем существующий канал
		0,				// атрибуты по умолчанию
		(HANDLE)NULL			// дополнительных атрибутов нет
```

```
	);

		// проверяем связь с каналом
	if (hNamedPipe == INVALID_HANDLE_VALUE)
	{
		cerr << "Connection with the named pipe failed." << endl
			<< "The last error code: " << GetLastError() << endl;
		cout << "Press any char to finish the client: ";
		cin >> c;
		return 0;
	}
		// пишем в именованный канал
	for (int i=0; i<10; i++)
	{
		DWORD dwBytesWritten;
		if (!WriteFile(
				hNamedPipe,		// дескриптор канала
				&i,			// данные
				sizeof(i),		// размер данных
				&dwBytesWritten,	// количество записанных байтов
				(LPOVERLAPPED)NULL	// синхронная запись
			))
		{
				// ошибка записи
			cerr << "Writing to the named pipe failed: " << endl
				<< "The last error code: " << GetLastError() << endl;
			cout << "Press any char to finish the client: ";
			cin >> c;
			CloseHandle(hNamedPipe);
			return 0;
		}
			// выводим число на консоль
		cout << "The number " << i << " is written to the named pipe." << endl;
		Sleep(1000);
	}
		// закрываем дескриптор канала
	CloseHandle(hNamedPipe);
		// завершаем процесс
	cout << "The data are written by the client." << endl
		<< "Press any char to finish the client: ";
	cin >> c;
	return 0;
}
```

Программа 6.2. Пример клиента именованного канала.

Теперь рассмотрим пример сервера именованного канала, который сначала создает именованный канал, затем ждет подключения к нему клиента. После этого сервер принимает от клиента одно сообщение, выводит это сообщение на консоль и посылает клиенту сообщение в ответ.

```
// Пример процесса сервера именованного канала.
// Сервер принимает сообщение от клиента и посылает ему сообщение в ответ.
// Внимание: в этом случае для работы в локальной сети вход  на клиентскую машину должен быть выполнен
// с тем же именем и паролем, что и на сервер.

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

int main()
{
	char	c;				// служебный символ
```

```
HANDLE          hNamedPipe;
char    lpszInMessage[80];                  // для сообщения от клиента
DWORD           dwBytesRead;                // для количества прочитанных байтов
char    lpszOutMessage[] = "The server has received a message.";    // обратное сообщение
DWORD           dwBytesWrite;               // для количества записанных байтов

        // создаем именованный канал для чтения
hNamedPipe = CreateNamedPipe(
        "\\\\.\\pipe\\demo_pipe",                   // имя канала
        PIPE_ACCESS_DUPLEX,                 // читаем из канала и пишем в канал
        PIPE_TYPE_MESSAGE | PIPE_WAIT,      // синхронная передача сообщений
        1,              // максимальное количество экземпляров канала
        0,              // размер выходного буфера по умолчанию
        0,              // размер входного буфера по умолчанию
        INFINITE,       // клиент ждет связь 500 мс
        (LPSECURITY_ATTRIBUTES)NULL     // защита по умолчанию
);
        // проверяем на успешное создание
if (hNamedPipe == INVALID_HANDLE_VALUE)
{
        cerr << "Creation of the named pipe failed." << endl
                << "The last error code: " << GetLastError() << endl;
        cout << "Press any char to finish server: ";
        cin >> c;
        return 0;
}

        // ждем, пока клиент свяжется с каналом
cout << "The server is waiting for connection with a client." << endl;
if(!ConnectNamedPipe(
                hNamedPipe,                     // дескриптор канала
                (LPOVERLAPPED)NULL          // связь синхронная
        ))
{
        cerr << "The connection failed." << endl
                << "The last error code: "<<GetLastError() << endl;
        CloseHandle(hNamedPipe);
        cout << "Press any char to finish the server: ";
        cin >> c;
        return 0;
}
        // читаем сообщение от клиента
if (!ReadFile(
                hNamedPipe,                     // дескриптор канала
                lpszInMessage,                  // адрес буфера для ввода данных
                sizeof(lpszInMessage),          // число читаемых байтов
                &dwBytesRead,                   // число прочитанных байтов
                (LPOVERLAPPED)NULL          // передача данных синхронная
        ))
{
        cerr << "Data reading from the named pipe failed." << endl
                << "The last error code: "<< GetLastError() << endl;
        CloseHandle(hNamedPipe);
        cout << "Press any char to finish the server: ";
        cin >> c;
        return 0;
}
        // выводим полученное от клиента сообщение на консоль
cout << "The server has received the following message from a client: "
        << endl << "\t" << lpszInMessage << endl;
        // отвечаем клиенту
if (!WriteFile(
                hNamedPipe,                     // дескриптор канала
                lpszOutMessage,                 // адрес буфера для вывода данных
```

```
                    sizeof(lpszOutMessage),         // число записываемых байтов
                    &dwBytesWrite,                  // число записанных байтов
                    (LPOVERLAPPED)NULL              // передача данных синхронная
                ))
        {
                cerr << "Data writing to the named pipe failed." << endl
                        << "The last error code: " << GetLastError() << endl;
                CloseHandle(hNamedPipe);
                cout << "Press any char to finish the server: ";
                cin >> c;
                return 0;
        }
                // выводим посланное клиенту сообщение на консоль
        cout << "The server send the following message to a client: "
                << endl << "\t" << lpszOutMessage << endl;
                // закрываем дескриптор канала
        CloseHandle(hNamedPipe);
                // завершаем процесс
        cout << "Press any char to finish the server: ";
        cin >> c;
        return 0;
}
```

Программа 6.3. Пример сервера именованного канала.

Обратим в этой программе внимание на следующий момент. Если клиент и сервер работают на разных компьютерах локальной сети, то вход как на компьютер сервера, так и на компьютер клиента, должен осуществляться с одинаковыми именами и паролями. Так как по умолчанию атрибуты защиты именованного канала устанавливаются таким образом, что он принадлежит только пользователю, создавшему этот именованный канал. В следующей программе мы установим атрибуты защиты таким образом, чтобы они разрешали доступ к именованному каналу любому пользователю.

```
// Пример процесса сервера именованного канала.
// Сервер принимает сообщение от клиента и посылает ему сообщение в ответ.
// В этом случае для работы в локальной сети вход на клиентскую машину может быть
// выполнен с любым именем и паролем.

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

int main()
{
        char    c;          // служебный символ
        SECURITY_ATTRIBUTES sa;             // атрибуты защиты
        SECURITY_DESCRIPTOR sd;             // дескриптор защиты
        HANDLE      hNamedPipe;
        char    lpszInMessage[80];          // для сообщения от клиента
        DWORD       dwBytesRead;            // для числа прочитанных байтов
        char    lpszOutMessage[] = "The server has received a message.";    // обратное сообщение
        DWORD       dwBytesWrite;           // для числа записанных байтов

                // инициализация атрибутов защиты
        sa.nLength = sizeof(sa);
        sa.bInheritHandle = FALSE;          // дескриптор канала ненаследуемый
                // инициализируем дескриптор защиты
        InitializeSecurityDescriptor(&sd,SECURITY_DESCRIPTOR_REVISION);
                // устанавливаем атрибуты защиты, разрешая доступ всем пользователям
        SetSecurityDescriptorDacl(&sd, TRUE, NULL, FALSE);
        sa.lpSecurityDescriptor = &sd;
                // создаем именованный канал для чтения
        hNamedPipe = CreateNamedPipe(
                "\\\\.\\pipe\\demo_pipe",           // имя канала
```

```
        PIPE_ACCESS_DUPLEX,                 // читаем из канала и пишем в канал
        PIPE_TYPE_MESSAGE | PIPE_WAIT,      // синхронная передача сообщений
        1,              // максимальное количество экземпляров канала
        0,              // размер выходного буфера по умолчанию
        0,              // размер входного буфера по умолчанию
        INFINITE,       // клиент ждет связь 500 мс
        &sa             // доступ для всех пользователей
);
        // проверяем на успешное создание
if (hNamedPipe == INVALID_HANDLE_VALUE)
{
        cerr << "Creation of the named pipe failed." << endl
                << "The last error code: " << GetLastError() << endl;
        cout << "Press any char to finish server: ";
        cin >> c;
        return 0;
}

        // ждем, пока клиент свяжется с каналом
cout << "The server is waiting for connection with a client." << endl;
if(!ConnectNamedPipe(
                hNamedPipe,                     // дескриптор канала
                (LPOVERLAPPED)NULL              // связь синхронная
        ))
{
        cerr << "The connection failed." << endl
                << "The last error code: " << GetLastError() << endl;
        CloseHandle(hNamedPipe);
        cout << "Press any char to finish the server: ";
        cin >> c;
        return 0;
}
        // читаем сообщение от клиента
if (!ReadFile(
                hNamedPipe,                     // дескриптор канала
                lpszInMessage,                  // адрес буфера для ввода данных
                sizeof(lpszInMessage),          // число читаемых байтов
                &dwBytesRead,                   // число прочитанных байтов
                (LPOVERLAPPED)NULL              // передача данных синхронная
        ))
{
        cerr << "Data reading from the named pipe failed." << endl
                << "The last error code: " << GetLastError() << endl;
        CloseHandle(hNamedPipe);
        cout << "Press any char to finish the server: ";
        cin >> c;
        return 0;
}
        // выводим полученное от клиента сообщение на консоль
cout << "The server has receivrd the following message from a client: "
        << endl << "\t" << lpszInMessage << endl;
        // отвечаем клиенту
if (!WriteFile(
                hNamedPipe,                     // дескриптор канала
                lpszOutMessage,                 // адрес буфера для вывода данных
                sizeof(lpszOutMessage),         // число записываемых байтов
                &dwBytesWrite,                  // число записанных байтов
                (LPOVERLAPPED)NULL              // передача данных синхронная
        ))
{
        cerr << "Data writing to the named pipe failed." << endl
                << "The last error code: " << GetLastError() << endl;
        CloseHandle(hNamedPipe);
        cout << "Press any char to finish the server: ";
```

```
		cin >> c;
		return 0;
	}
		// выводим посланное клиенту сообщение на консоль
	cout << "The server send the following message to a client: "
		<< endl << "\t" << lpszOutMessage << endl;
		// закрываем дескриптор канала
	CloseHandle(hNamedPipe);
		// завершаем процесс
	cout << "Press any char to finish the server: ";
	cin >> c;
	return 0;
}
```

Программа 6.4. Пример сервера именованного канала.

Теперь приведем пример клиента именованного канала, который вводит сначала с консоли имя компьютера в локальной сети, на котором запущен сервер именованного канала. Затем связывается с этим именованным каналом. После этого клиент передает серверу одно сообщение и получает от него сообщение в ответ, которое выводит на консоль.

```
// Пример процесса клиента именованного канала.

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

int main()
{
	char	c;			// служебный символ
	HANDLE	hNamedPipe;
	char	machineName[80];
	char	pipeName[80];
	char	lpszOutMessage[]="How do you do server?";	// сообщение серверу
	DWORD	dwBytesWritten;		// для числа записанных байтов
	char	lpszInMessage[80];		// для сообщения от сервера
	DWORD	dwBytesRead;		// для числа прочитанных байтов

		// вводим имя машины в сети, на которой работает сервер
	cout << "Enter a name of the server machine: ";
	cin >> machineName;
		// подставляем имя машины в имя канала
	wsprintf(pipeName, "\\\\%s\\pipe\\demo_pipe",
		machineName);

		// связываемся с именованным каналом
	hNamedPipe = CreateFile(
		pipeName,				// имя канала
		GENERIC_READ | GENERIC_WRITE,		// читаем и записываем в канал
		FILE_SHARE_READ | FILE_SHARE_WRITE,	// разрешаем чтение и запись в канал
		(LPSECURITY_ATTRIBUTES) NULL,		// защита по умолчанию
		OPEN_EXISTING,				// открываем существующий канал
		FILE_ATTRIBUTE_NORMAL,			// атрибуты по умолчанию
		(HANDLE)NULL				// дополнительных атрибутов нет
		);

		// проверяем связь с каналом
	if (hNamedPipe==INVALID_HANDLE_VALUE)
	{
		cerr << "Connection with the named pipe failed." << endl
			<< "The last error code: " << GetLastError() << endl;
		cout << "Press any char to finish the client: ";
		cin >> c;
		return 0;
```

```
	}
		// пишем в именованный канал
	if (!WriteFile(
		hNamedPipe,		// дескриптор канала
		lpszOutMessage,		// данные
		sizeof(lpszOutMessage),	// размер данных
		&dwBytesWritten,	// количество записанных байтов
		(LPOVERLAPPED)NULL		// синхронная запись
		))
	{
			// ошибка записи
		cerr << "Writing to the named pipe failed: " << endl
			<< "The last error code: " << GetLastError() << endl;
		cout << "Press any char to finish the client: ";
		cin >> c;
		CloseHandle(hNamedPipe);
		return 0;
	}
		// выводим посланное сообщение на консоль
	cout << "The client has send the following message to a server: "
		<< endl << "\t" << lpszOutMessage << endl;
		// читаем из именованного канала
	if (!ReadFile(
		hNamedPipe,		// дескриптор канала
		lpszInMessage,		// данные
		sizeof(lpszInMessage),	// размер данных
		&dwBytesRead,		// количество записанных байт
		(LPOVERLAPPED)NULL		// синхронная запись
		))
	{
			// ошибка записи
		cerr << "Reading to the named pipe failed: " << endl
			<< "The last error code: " << GetLastError() << endl;
		cout << "Press any char to finish the client: ";
		cin >> c;
		CloseHandle(hNamedPipe);
		return 0;
	}
		// выводим полученное сообщение на консоль
	cout << "The client has received the following message from a server: "
		<< endl << "\t" << lpszInMessage << endl;
		// закрываем дескриптор канала
	CloseHandle(hNamedPipe);
		// завершаем процесс
	cout << "Press any char to finish the client: ";
	cin >> c;
	return 0;
}
```

Задача.

Процессы должны работать на разных компьютерах!

6.1. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя имя консоли процесса **Client**, цвет всего фона (15 цветов) консоли **Client**.
- Передаёт данные процессу-клиенту.
- При нажатии клавиши мыши передаёт **Client** , следующий цвет в палитре.
- Закончить работу, после нажатия клавиши.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о цвете.
- Устанавливает фон.

6.2. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя координаты и размер курсора консоли.
- Запрашивает строку символов.
- При нажатии клавиши мыши печатает текст с позиции курсора мыши и передаёт новые координаты процессу-клиенту.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные об координатах и размере курсора консоли, строку.
- Печатает строку с переданной позиции.

6.3. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя размер окна консоли и цвет фона.
- Запрашивает строку символов.
- Заполняет буфер экрана введённым символом и закрашивает фон цветом.
- При нажатии клавиши мыши передаёт данные процессу-клиенту.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о размере и цвете окна, строку символов.
- Заполняет буфер экрана строкой символов.

6.4. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя цвет выводимых символов, координаты вывода символов.
- Запрашивает строку.
- Передает эти параметры процессу-клиенту, который запущен на другом компьютере в локальной сети.
- При нажатии клавиши мыши передаёт сообщение **Client**, что нужно всё стереть с экрана

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера начальные данные о цвете символов, положении курсора в окне консоли.
- Устанавливает курсор и выводит строку с заданной позиции.
- Стирает всё с экрана, если получено сообщение от сервера.

6.5. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя цвет фона консоли, цвет выводимых символов, координаты вывода символов.
- Запрашивает строку.
- Передает эти параметры процессу-клиенту, который запущен на другом компьютере в локальной сети.
- При нажатии левой клавиши мыши передаёт сообщение **Client**, что нужно изменить цвет символов экрана

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера начальные данные о цвете символов, положении курсора в окне консоли.
- Выводит строку с заданной позиции.
- Меняет цвет выведенных символов на экране, если получено сообщение от сервера.

6.6. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя цвет фона консоли, цвет выводимых символов, координаты вывода символов.
- Запрашивает строку.
- Передает эти параметры процессу-клиенту, который запущен на другом компьютере в локальной сети.
- При двойном нажатии левой клавиши мыши передаёт сообщение **Client**, что нужно изменить цвет экрана
- При нажатии правой клавиши мыши передаёт сообщение **Client**, что нужно вывести символы с новой позиции (позиция курсора мыши)

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера начальные данные о цвете символов, положении курсора в окне консоли.
- Выводит строку с заданной позиции.
- Меняет цвет экрана либо выводит символы с новой позиции, если получено соответствующее сообщение от сервера.

6.7. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя размер, цвет и начальное положение прямоугольника в окне консоли, число N.
- Запрашивает символы для заполнения прямоугольника.
- Отображает прямоугольник.
- При двойном нажатии правой клавиши мыши рисует прямоугольник другим цветом (выбрать случайным образом), передает данные о прямоугольнике процессу-серверу.
- При нажатии левой клавиши мыши – завершение работы процессов.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о размере, цвете, символе заполнителе и положении прямоугольника в окне консоли.
- Отображает прямоугольник.

6.8. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя размер, цвет и начальное положение прямоугольника в окне консоли, число N.
- Запрашивает символы для заполнения прямоугольника.

- Отображает прямоугольник.
- При нажатии правой клавиши мыши увеличивает размер прямоугольника, передает данные о прямоугольниках процессу-серверу.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о размере, цвете, символе заполнителе и положении прямоугольника в окне консоли.
- Увеличивает либо уменьшает прямоугольник в N раз.

6.9. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя строку символов, цвет выводимых символов, координаты вывода символов, размер буфера для заполнения строчками символов.
- Передает эти параметры процессу-клиенту, который запущен на другом компьютере в локальной сети.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера начальные данные о строке, размере, цвете символов, положении курсора в окне консоли и размер буфера для заполнения символами.
- Устанавливает курсор и заполняет буфер введённой строкой с заданной позиции.

6.10. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя, размер, цвет прямоугольника в окне консоли, числа N и M.
- Запрашивает, размер и массив символов для заполнения прямоугольников.
- Отображает прямоугольник.
- Через интервалы времени M, увеличивает размер прямоугольника на число N, и отображает прямоугольник правее (или с другой свободной стороны) от предыдущего
- При нажатии клавиши мыши последовательно передает данные о прямоугольниках процессу-серверу.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о размере, цвете, символе заполнителе и положении прямоугольника в окне консоли.
- Отображает прямоугольник.

6.11. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя количество, размер, цвет и начальное положение прямоугольников в окне консоли.
- Рисует прямоугольники.
- При установке курсора мыши и нажатии клавиши мыши на одном из прямоугольников окне процесса-сервера или в окне процесса-клиента.
- Передает данные о размере, цвете и положении прямоугольника процессу-клиенту.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера начальные данные о размере, цвете и положении прямоугольника в окне консоли.
- Рисует прямоугольник.

6.12. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя количество, размер, цвет символов-заполнителей и начальное положение прямоугольников в окне консоли.
- Запрашивает символы для заполнения прямоугольников.
- Отображает прямоугольники.
- При двойном нажатии правой клавиши мыши последовательно передает данные о прямоугольниках процессу-серверу.
- При нажатии левой клавиши мыши последовательно стирает прямоугольники, и Процесс-клиент тоже стирает прямоугольники.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о размере, цвете, символе заполнителе и положении прямоугольника в окне консоли.
- Отображает либо стирает прямоугольник.

6.13. Написать программы двух консольных процессов Server, Client, работающих на разных компьютерах в локальной сети.

Процесс- **Server**, который выполняет следующие действия.

- Запрашивает у пользователя размер окна консоли и цвет фона.
- Запрашивает строку символов.
- Заполняет буфер экрана введённым символом и закрашивает фон цветом.
- При нажатии клавиши мыши передаёт данные процессу- **Client**.

Процесс- **Client**, который выполняет следующие действия.

- Получает от процесса сервера данные о размере и цвете окна, строку символов.
- Заполняет буфер экрана строкой символов.

ВГКС
Кафедра ПОСТ
Курс «Системное программное обеспечение»
Лабораторная работа №6 (4 часа)
Группа №_____

№	Фамилия Имя Отчество	Вариант	Дата сдачи	Примечания
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				