**«Санкт-Петербургский Государственный Политехнический Университет»**

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №6**

Тема: «Средства межпроцессного взаимодействия в ОС Windows»

Дисциплина: «Операционные системы»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выполнил студент гр.3530901/70203 | \_\_\_\_\_\_\_\_\_\_\_ | | | Иванов И.Д. |
| Преподав­атель, к.т.н., доц. | \_\_\_\_\_\_\_\_\_\_\_ | | | Душутина Е.В. |
|  |  | “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. | | |
|  |  | |  | |

**1. Цель работы**

Изучение средств межпроцессного взаимодействия (IPC) в ОС Windows.

**2. Программа работы**

1.1. Создать клиент-серверное приложение, позволяющее набираемые символы в терминальном окне командной строки (сервер) отображать в окно процесса-потомка (клиент).

1.2. Создать эхо-сервер, взаимодействующий с клиентом посредством pipe.

2.1. Реализовать между одним клиентом и сервером обмен данными, вводимыми с консоли на стороне клиента и возвращаемыми сервером обратно до получения команды exit.

2.2. Реализовать между сервером и множеством клиентов обмен данными, вводимыми с консоли на стороне клиента и возвращаемыми сервером обратно до получения команды exit.

2.3. Модифицировать приложение из пункта 2.2 для сетевого обмена информацией.

3.1. Разработать программу локального обмена сокетами с использованием потокового протокола с установлением соединения (TCP в стеке TCP/IP).

3.2. Модифицировать программу для локального обмена с множеством клиентов и с доступом к общему ресурсу.

4.1. Задание обработчика сигналов завершения для консольного приложения.

4.2. Предложить собственную реализацию обработчика сигнала.

5.1. Создать программу, в которой первый процесс генерирует случайное число и записывает его в буфер, доступный второму процессу, откуда он его и считывает с последующим выводом.

6.1. Предложить собственную реализацию приложения, иллюстрирующую обмен информацией почтовыми слотами.

6.2. Продемонстрировать возможность локального и удаленного доступа.

**Система, на которой производится работа:**

C:\Windows\system32>ver

Microsoft Windows [Version 10.0.14393]

C:\Windows\system32>g++ --version

g++ (MinGW.org GCC-8.2.0-3) 8.2.0

Кол-во ядер: 4

**3. Выполнение работы**

**1. Неименованные каналы (Pipe)**

Посредством pipe-канала можно передавать данные только между двумя процессами. В основе взаимодействия лежит так называемая файловая модель функционирования. Один из процессов создает канал, другой открывает его. После этого оба процесса могут передавать данные через канал в одну или обе стороны, используя для этого функции, предназначенные для работы с файлами, такие как ReadFile и WriteFile.

Анонимные каналы (anonymous channels) Windows обеспечивают однонаправленное (полудуплексное) посимвольное межпроцессное взаимодействие. Каждый канал имеет два дескриптора: дескриптор чтения (read handle) и дескриптор записи (write handle).

Функция, с помощью которой создаются анонимные каналы, имеет

следующий прототип:

BOOL CreatePipe(PHANDLE phRead, PHANDLE phWrite, LPSECURITY\_ATTRIBUTES lpsa, DWORD nsize)

PHANDLE phRead, PHANDLE phWrite – указатели на дескрипторы чтения и записи соответственно;

LPSECURITY\_ATTRIBUTES lpsa – атрибуты защиты (там указывается и флаг разрешения наследования дескриптора);

DWORD nsize – размер канала в байтах. Носит рекомендательный характер (конкретный размер будет определен системой). При значении равном 0 используется размер по-умолчанию.

Возвращаемое значение: 0 – при ошибке, не 0 при нормальной работе.

Для получения данных из канала используется функция:

BOOL WINAPI ReadFile(HANDLE hFile, LPVOID lpBuffer,DWORD NumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);

HANDLE hFile – дескриптор файла, из которого производится чтение;

LPVOID lpBuffer – указатель на буфер-приемник данных;

DWORD nNumberOfBytesToRead – максимальное число читаемых байт (размер буфера);

LPDWORD lpNumberOfBytesRead – указатель на переменную, в которую заносится число фактически прочитанных байт. Этот параметр может быть равен 0, если в момент чтения указатель находился на конце файла либо если функция завершилась с ошибкой;

LPOVERLAPPED lpOverlapped – специальная структура, позволяющая организовать асинхронный ввод-вывод. Она содержит в своем составе позицию, с которой будет производиться операция в файле и событие (его дескриптор),

которое будет переходить в сигнальное состояние по завершению выполнения операции (сброс ручной).

Возвращаемое значение: в случае успешного выполнения (которое считается таковым, даже если не был считан ни один байт из-за попытки чтения с выходом за пределы файла) — TRUE, иначе — FALSE.

Функция возвращает управление в следующих ситуациях: заверена операция записи на дескрипторе записи канала, прочитано требуемое количество данных или в случае возникновения ошибки.

Для записи данных в канал используется функция:

BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer,DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)

Все параметры аналогичны функции ReadFile (только вместо чтения производится запись и указывается буфер, из которого будут забираться данные).

Возвращаемое значение: в случае успешного выполнения — TRUE, иначе — FALSE.

Если функция возвращает управление, это еще не означает, что данные передались на диск, если только файл не был создан с флагом FILE\_FLAG\_WRITE\_THROUGH.

Функция не вернет управления, пока все байты не будут записаны (т.е. пока в буфере не будет достаточно для них места).

После создания канала необходимо передать клиентскому процессу его дескрипторы (или один из них), что обычно делается с помощью механизма наследования.

Напомним, что для наследования описателя нужно, чтобы процесс потомок создавался функцией CreateProcess с флагом наследования TRUE. Предварительно нужно создать наследуемые описатели. Сделать это можно несколькими способами.

Например, путем явной спецификации параметра bInheritHandle структуры SECURITY\_ATTRIBUTES при создании канала. Поскольку процессу-потомку значение наследуемого дескриптора пока еще не известно, родительский процесс должен передать это значение потомку либо через механизм межпроцессного взаимодействия (Interprocess Communication, IPC), либо путем назначения дескриптора стандартному устройству ввода/вывода в структуре STARTUPINFO. Последний вариант позволяет произвести передачу дескрипторов без внесения изменений в дочернюю программу.

Другим способом является создание наследуемого дубликата имеющегося описателя при помощи функции DuplicateHandle и последующая передача его создаваемому процессу через командную строку или каким-либо иным образом (через IPC).

**1.1. Создать клиент-серверное приложение, позволяющее набираемые символы в терминальном окне командной строки (сервер) отображать в окно процесса-потомка (клиент).**

В программе-сервере master создается неименованный канал для связи с процессом-потомком, порождается сам процесс-потомок (программа-клиент) slave. На стороне сервера производится запись из консоли в канал. В slave открывается неименованный канал и осуществляется считывание из него в новое окно. Запись\чтение канала производится с помощью стандартных потоков std\_in и std\_out.

**Исходный код сервера:**

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <iostream>

using namespace std;

int main()

{ // инициализируем необходимые структуры

STARTUPINFO si = {sizeof(si)};

SECURITY\_ATTRIBUTES sa;

PROCESS\_INFORMATION pi;

char buf[1024];

char t= '\n';

HANDLE newstdread, newstdwrite;//хэндлы потоков для пайпа

//инициализируем нужные поля SECURITY\_ATTRIBUTES

sa.nLength = sizeof(sa);

sa.lpSecurityDescriptor = NULL;

sa.bInheritHandle = true; //разрешаем наследование дескрипторов

//создаем анонимный канал(создаем пайп для stdin)

if (!CreatePipe(&newstdread, //указатель на пеpеменную типа dword,

//котоpая получит хэндл конца чтения пайпа

&newstdwrite, //указатель на пеpеменную типа dword, котоpая получит хэндл на конец записи пайпа

&sa, // указатель на структуру атрибутов безопасности

0)) //размер буфера, используется по умолчанию

{

cout << "I can't CreatePipe";

getch();

return 0;

}

else

cout << "\nPipe Created!\n";

//выводим на экран дескриптор потока ввода анонимного канала

cout << "The read HANDLE of PIPE = " << newstdread << endl;

//обнуляем поля STARTUPINFO и задаем нужные значения

ZeroMemory(&si, sizeof(STARTUPINFO));

si.cb = sizeof(STARTUPINFO);

si.dwFlags = STARTF\_USESTDHANDLES | STARTF\_USESHOWWINDOW;

si.wShowWindow = SW\_NORMAL;

//подменяем стандартный дескриптор ввода дескриптором ввода канала

si.hStdInput = newstdread;

si.hStdOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);

si.hStdError = si.hStdOutput;

TCHAR czCommandLine[] = "C:\\lab\_6\\1\_1\\slave1.exe";;

if (!CreateProcess(NULL, czCommandLine, NULL, NULL, TRUE,

CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi))

{

cout << "Error: Сan't CreateProcess";

getch();

CloseHandle(newstdread);

CloseHandle(newstdwrite);

return 0;

}

else

cout << "\nProcess Created!!!\n";

memset(buf, '\0', sizeof(buf));

cout << buf;

unsigned long bread;

cout << "STD INPUT HANDLE = " <<

GetStdHandle(STD\_INPUT\_HANDLE) << endl;

cout << "STD OUTPUT HANDLE = " <<

GetStdHandle(STD\_OUTPUT\_HANDLE) << endl;

while(1)

{ memset(buf, '\0', sizeof(buf));

\*buf = (char)getch();

cout.put(\*buf);

if(\*buf==13)

{

\*buf = '\n';

cout.put(\*buf);

}

WriteFile(newstdwrite, //указатель на пишущих хэндл канала

buf, // указатель на буфер

1, //кол-во байт данных,записываемых в буфер

&bread, // указатель на переменную, хранящую кол-во байт,

//записанных в буфер

NULL); //т.к. 1-й аргумент не был открыт с флагом

FILE\_FLAG\_OVERLAPPED;

if(\*buf==27)

break;

}

TerminateProcess(pi.hProcess,0); // завершение процесса

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

CloseHandle(newstdread);

CloseHandle(newstdwrite);

system("PAUSE");

return 0;}

**Исходный код клиента:**

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include <windows.h>

using namespace std;

int main()

{

char buf[2];

unsigned long avail;

cout << "STD INPUT HANDLE = " <<

GetStdHandle(STD\_INPUT\_HANDLE) << "\n";

cout << "STD OUTPUT HANDLE = "<<

GetStdHandle(STD\_OUTPUT\_HANDLE) << "\n";

unsigned long bread;

while(1){

PeekNamedPipe(

GetStdHandle(STD\_INPUT\_HANDLE),

NULL,

NULL,

NULL,

&avail,

NULL);

if(avail)

{ memset(buf, '\0', sizeof(buf));

ReadFile(

GetStdHandle(STD\_INPUT\_HANDLE), buf, 1, &bread, NULL);

cout << buf;

}

}

return 0;

}

**Результат выполнения программы:**

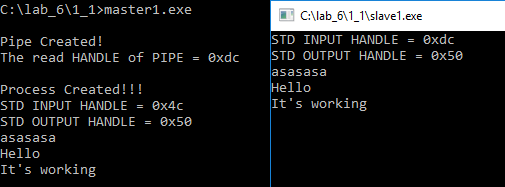
****

Рис.1. Результаты работы приложения с IPC pipe и консолью

Видно, что после запуска программы-сервера в командной строке была сначала выведена строка об успешном создании пайпа. Затем был показан дескриптор потока ввода анонимного канала, после чего вывелось сообщение об успешном создании процесса и в новом окне порождён процесс slave1.exe, после чего все введённые в первом терминале сообщения сразу же отображались и в новом окне клиента.

Unnamed pipe является однонаправленным локальным средством взаимодействия процессов с наследованием или родственных, как в данном случае.

**1.2. Создать эхо-сервер, взаимодействующий с клиентом посредством pipe.**

В программе используется передача дескрипторов через наследование. По причине того, что анонимный канал является полудуплексным, для организации эхо-сервера необходимо создавать 2 канала (для передачи от клиента серверу и обратно). При этом ненужные дескрипторы каналов закрываются только на стороне сервера (т.к. клиент наследует 4 дескриптора, а явно мы передаем только 2 дескриптора).

Дескрипторы каналов связываются со стандартным вводом и выводом клиентского процесса. Поэтому клиент выводит информацию в поток ошибок (что приведет к выводу в консоль процесса-клиента).

Клиент передает сообщение, например, вида: «message num 1». Сервер передает данное сообщение обратно. Процессы завершаются после передачи 10 сообщений.

**Исходный код сервера:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <stdio.h>

int main(int argc, char\* argv[]) {

HANDLE hReadPipeFromServToClient, hWritePipeFromServToClient;

//дескрипторы канала для

//передачи от сервера клиенту

HANDLE hReadPipeFromClientToServ, hWritePipeFromClientToServ;

//дескрипторы канала для

//передачи от сервера клиенту

SECURITY\_ATTRIBUTES PipeSA = {sizeof(SECURITY\_ATTRIBUTES), NULL, TRUE};

//чтобы сделать

//дескрипторы наследуемыми

//создаем канал для передачи от сервера клиенту, сразу делаем дескрипторы наследуемыми

if(CreatePipe(&hReadPipeFromServToClient,&hWritePipeFromServToClient,&PipeS A,0)==0) {

printf("impossible to create anonymous pipe from serv to client\n");

getchar();

return 1000;

}

//создаем канал для передачи от клиента серверу, сразу делаем дескрипторы наследуемыми

if(CreatePipe(&hReadPipeFromClientToServ,&hWritePipeFromClientToServ,&PipeS A,0)==0) {

printf("impossible to create anonymous pipe from client to serv\n");

getchar();

return 1001;

}

PROCESS\_INFORMATION processInfo\_Client;

// информация о процессе-клиенте

STARTUPINFO startupInfo\_Client;

//структура, которая описывает внешний вид основного

//окна и содержит дескрипторы стандартных устройств нового процесса, используем для установки

//процесс-клиент будет иметь те же параметры запуска, что и сервер, за исключением

//дескрипторов ввода, вывода и ошибок

GetStartupInfo(&startupInfo\_Client);

startupInfo\_Client.hStdInput = hReadPipeFromServToClient; //устанавливаем поток ввода

startupInfo\_Client.hStdOutput=hWritePipeFromClientToServ; //установим поток вывода

startupInfo\_Client.hStdError=GetStdHandle(STD\_ERROR\_HANDLE); //установим поток ошибок

startupInfo\_Client.dwFlags = STARTF\_USESTDHANDLES; //устанавливаем наследование

//создаем процесс клиента

TCHAR czCommandLine[] = "C:\\lab\_6\\1\_2\\PipesSonProc.exe";

CreateProcess(NULL, czCommandLine, NULL, NULL, TRUE, CREATE\_NEW\_CONSOLE, NULL, NULL, &startupInfo\_Client, &processInfo\_Client);

CloseHandle(processInfo\_Client.hThread);

//закрываем дескрипторы созданного процесса и его

//потока

CloseHandle(processInfo\_Client.hProcess);

//закрываем ненужные дескрипторы каналов, которые не использует сервер

CloseHandle(hReadPipeFromServToClient);

CloseHandle(hWritePipeFromClientToServ);

#define BUF\_SIZE 100 //размер буфера для сообщений

BYTE buf[BUF\_SIZE]; //буфер приема/передачи

DWORD readbytes,writebytes; //число прочитанных/переданных байт

for(int i=0;i<10;i++) { //читаем данные из канала от клиента

if(!ReadFile(hReadPipeFromClientToServ,buf,BUF\_SIZE,&readbytes,NULL)) {

printf("impossible to use readfile\n GetLastError= %d\n",GetLastError());

getchar();

return 10000;

}

printf("get from client: \"%s\"\n",buf);

if(!WriteFile(hWritePipeFromServToClient,buf,readbytes,&writebytes,NULL)) {

printf("impossible to use writefile\n GetLastError= %d\n",GetLastError());

getchar();

return 10001;

} //пишем данные в канал клиенту

}

//закрываем HANDLE каналов

CloseHandle(hReadPipeFromClientToServ);

CloseHandle(hWritePipeFromServToClient);

printf("server ended work\n Press any key");

getchar();

return 0;

}

**Исходный код клиента:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int main(int argc, char\* argv[]) {

char strtosend[100]; //строка для передачи

char getbuf[100]; //буфер приема

int bytestosend; //число передаваемых байт

DWORD bytessended,bytesreaded; //число переданных и принятых байт

for(int i=0;i<10;i++) {

bytestosend=sprintf(strtosend,"message num %d",i+1);

//формирование строки для

//передачи

strtosend[bytestosend]=0;

fprintf(stderr,"client sended: \"%s\"\n",strtosend);

if(!WriteFile(GetStdHandle(STD\_OUTPUT\_HANDLE),strtosend,bytestosend+1,&bytesreaded,NULL)) {

fprintf(stderr,"Error with writeFile\n Wait 5 sec GetLastError= %d\n",GetLastError());

Sleep(5000);

return 1000;

}

if (!ReadFile(GetStdHandle(STD\_INPUT\_HANDLE), getbuf, 100, &bytesreaded, NULL)) {

fprintf(stderr,"Error with readFile\n Wait 5 sec GetLastError= %d\n",GetLastError());

Sleep(5000);

return 1001;

}

fprintf(stderr,"Get msg from server: \"%s\"\n",getbuf);

}

fprintf(stderr,"client ended work\n Wait 5 sec");

Sleep(5000);

return 0;

}

**Результат выполнения программы:**

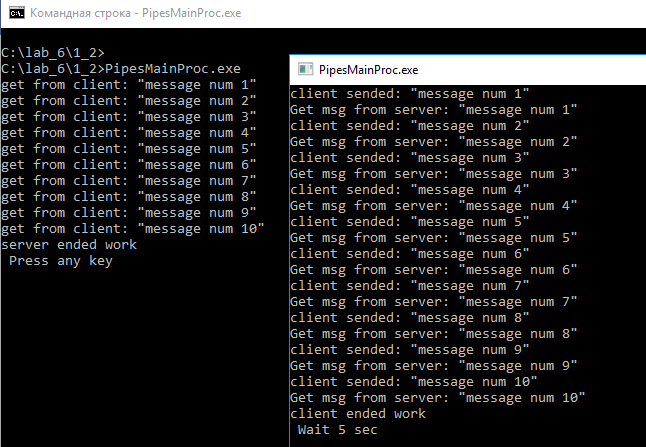


Рис.2. Результат работы приложения с эхо-сервером на основе pipe

**2. Именованные каналы**

Именованные каналы являются дуплексными, ориентированы на обмен сообщениями и обеспечивают взаимодействие через сеть. Кроме того, один именованный канал может иметь несколько открытых дескрипторов. В сочетании с удобными, ориентированными на выполнение транзакций функциями эти возможности делают именованные каналы пригодными для создания клиент-серверных систем. Обмен данными может быть синхронным и асинхронным.

Для создания именованного канала используется функция:

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

LPCTSTR lpName - указатель на имя канала, который должен иметь следующую форму:

«\\.\pipe\[path]pipename»

Точка (.) обозначает локальный компьютер; таким образом, создать канал на удаленном компьютере невозможно;

DWORD dwOpenMode – задает режим, в котором открывается канал. Может иметь следующие значения (которые могут комбинироваться):

• PIPE\_ACCESS\_INBOUND - только для чтения (передача от клиента-серверу, сервер открывает pipe с параметром чтения, клиент – записи);

• PIPE\_ACCESS\_OUTBOUND - только для записи;

• PIPE\_ACCESS\_DUPLEX - для чтения и записи;

• FILE\_FLAG\_OVERLAPPED - использование асинхронных операций (ввод и вывод с перекрытием). Данный режим позволяет процессу выполнять полезную работу параллельно с проведением операций в канале;

• FILE\_FLAG\_WRITE\_THROUGH - в этом режиме функции, работающие с каналом, не возвращают управление до тех пор, пока не будет полностью завершена операция на удаленном компьютере. Используется только с каналом, ориентированном на передачу отдельных байт и только в том случае, когда канал создан между процессами, запущенными на различных станциях сети;

DWORD dwPipeMode – имеются три пары взаимоисключающих значений этого параметра. Эти значения указывают, ориентирована ли запись на работу с сообщениями или байтами, ориентировано ли чтение на работу с сообщениями или блоками, и блокируются ли операции чтения.

• PIPE\_TYPE\_BYTE и PIPE\_TYPE\_MESSAGE — указывают, соответственно, должны ли данные записываться в канал как поток байтов или как сообщения. Для всех экземпляров каналов с одинаковыми именами следует использовать одно и то же значение.

• PIPE\_READMODE\_BYTE и PIPE\_READMODE\_MESSAGE — указывают, соответственно, должны ли данные считываться как поток байтов или как сообщения. Значение PIPE\_READMODE\_MESSAGE требует использования значения PIPE\_TYPE\_MESSAGE.

• PIPE\_WAIT и PIPE\_NOWAIT — определяют, соответственно, будет или не будет блокироваться операция ReadFile. Следует использовать значение PIPE\_WAIT, поскольку для обеспечения асинхронного ввода/вывода существуют лучшие способы.

DWORD nMaxInstances – определяет количество экземпляров каналов, а следовательно, и количество одновременно подключенных клиентов;

DWORD nOutBufferSize, DWORD nInBufferSize – позволяют указать размеры в байтах входного и выходного буферов канала. Если равны 0, то используются размер по-умолчанию;

DWORD nDefaultTimeOut – длительность интервала ожидания по-умолчанию для функции WaitNamedPipe;

LPSECURITY\_ATTRIBUTES lpSecurityAttributes – атрибуты защиты.

Возвращаемое значение: в случае успеха возвращается дескриптор канала со стороны сервера, в случае неудачи - INVALID\_HANDLE\_VALUE.

При первом вызове функции CreateNamedPipe происходит создание самого именованного канала, а не просто его экземпляра. Закрытие последнего открытого дескриптора экземпляра именованного канала приводит к уничтожению этого экземпляра (обычно существует по одному дескриптору на каждый экземпляр). Уничтожение последнего экземпляра именованного канала приводит к уничтожению самого канала, в результате чего имя канала становится вновь доступным для повторного использования.

После создания именованного канала сервер может ожидать подключения клиента, вызывая функцию:

BOOL ConnectNamedPipe(HANDLE hNamedPipe, LPOVERLAPPED lpOverlapped);

HANDLE hNamedPipe – дескриптор именованного канала;

LPOVERLAPPED lpOverlapped – указатель на структуру OVERLAPPED.

Возвращаемое значение – в случае успеха –TRUE, в случае неудачи – FALSE.

Функция возвращает управление после соединения с клиентом (если второй параметр функции равен NULL). После сервер обменивается с клиентом данными при помощи функций WriteFile и ReadFile.

По завершению обмена сервер вызывает функцию:

BOOL DisconnectNamedPipe(HANDLE hNamedPipe);

HANDLE hNamedPipe – дескриптор именованного канала.

Возвращаемое значение – в случае успеха – TRUE, в случае неудачи – FALSE.

Для подключения клиента к именованному каналу применяется функция CreateFile, при вызове которой указывается имя именованного канала:

HANDLE CreateFile( LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY\_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDistribution, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile);

LPCTSTR lpFileName – имя канала. Часто клиент и сервер выполняются на одном компьютере, и в этом случае для указания имени канала используется следующая форма:

«\\.\pipe\[path]pipename»

Если сервер находится на другом компьютере, для указания имени канала используется следующая форма: «\\servername\pipe\[path]pipename»

Использование точки (.) вместо имени локального компьютера в случае, когда сервер является локальным, позволяет значительно сократить время подключения;

DWORD dwDesiredAccess – тип доступа. Можно использовать комбинацию из следующих констант:

• 0 - доступ запрещен, однако приложение может определять атрибуты файла, канала или устройства, открываемого при помощи функции CreateFile XE "CreateFile";

• GENERIC\_READ - разрешен доступ на чтение из файла или канала Pipe;

• GENERIC\_WRITE - разрешен доступ на запись в файл или канал Pipe;

DWORD dwShareMode – с помощью данного параметра задаются режимы совместного использования, открываемого или создаваемого файла. Можно указать комбинацию следующих констант:

• 0 - совместное использование файла запрещено;

• FILE\_SHARE\_READ - другие приложения могут открывать файл с помощью функции CreateFile для чтения;

• FILE\_SHARE\_WRITE - аналогично предыдущему, но на запись; LPSECURITY\_ATTRIBUTES lpSecurityAttributes – атрибуты защиты;

DWORD dwCreationDistribution - определяет действия, выполняемые функцией CreateFile, если приложение пытается создать файл, который уже существует. Могут быть указаны следующие константы:

• CREATE\_NEW - если создаваемый файл уже существует, функция CreateFile возвращает код ошибки;

• CREATE\_ALWAYS - существующий файл перезаписывается, при этом содержимое старого файла теряется;

• OPEN\_EXISTING - открывается существующий файл. Если файл с указанным именем не существует, функция CreateFile возвращает код ошибки;

• OPEN\_ALWAYS - если указанный файл существует, он открывается. Если файл не существует, он будет создан;

• TRUNCATE\_EXISTING - если файл существует, он открывается, после чего длина файла устанавливается равной нулю. Содержимое старого файла теряется. Если же файл не существует, функция CreateFile возвращает код ошибки;

DWORD dwFlagsAndAttributes – задает атрибуты и флаги для файла. В качестве атрибутов могут быть указаны следующие:

• FILE\_ATTRIBUTE\_ARCHIVE - файл был архивирован (выгружен);

• FILE\_ATTRIBUTE\_COMPRESSED - файл, имеющий этот атрибут, динамически сжимается при записи и восстанавливается при чтении. Если этот атрибут имеет каталог, то для всех расположенных в нем файлов и каталогов также выполняется динамическое сжатие данных;

• FILE\_ATTRIBUTE\_NORMAL - остальные перечисленные в этом списка атрибуты не установлены;

• FILE\_ATTRIBUTE\_HIDDEN - скрытый файл;

• FILE\_ATTRIBUTE\_READONLY - файл можно только читать;

• FILE\_ATTRIBUTE\_SYSTEM - файл является частью операционной системы.

В качестве флагов можно указать следующие:

• FILE\_FLAG\_WRITE\_THROUGH - отмена промежуточного кэширования данных для уменьшения вероятности потери данных при аварии;

• FILE\_FLAG\_NO\_BUFFERING - отмена промежуточной буферизации или кэширования. При использовании этого флага необходимо выполнять чтение и запись порциями, кратными размеру сектора (обычно 512 байт);

• FILE\_FLAG\_OVERLAPPED - асинхронное выполнение чтения и записи. Во время асинхронного чтения или записи приложение может продолжать обработку данных;

• FILE\_FLAG\_RANDOM\_ACCESS - указывает, что к файлу будет выполняться произвольный доступ. Флаг предназначен для оптимизации кэширования;

• FILE\_FLAG\_SEQUENTIAL\_SCAN - указывает, что к файлу будет выполняться последовательный доступ от начала файла к его концу. Флаг предназначен для оптимизации кэширования;

• FILE\_FLAG\_DELETE\_ON\_CLOSE - файл будет удален сразу после того как приложение закроет его идентификатор. Этот флаг удобно использовать для временных файлов;

• FILE\_FLAG\_BACKUP\_SEMANTICS - файл будет использован для выполнения операции выгрузки или восстановления. При этом выполняется проверка прав доступа;

• FILE\_FLAG\_POSIX\_SEMANTICS - доступ к файлу будет выполняться в соответствии со спецификацией POSIX;

HANDLE hTemplateFile – предназначен для доступа к файлу шаблона с расширенными атрибутами создаваемого файла.

Возвращаемое значение: в случае успешного завершения функция возвращает идентификатор открытого файла/канала, в случае неудачи - INVALID\_HANDLE\_VALUE.

С помощью функции WaitNamedPipe процесс может выполнять ожидание момента, когда канал Pipe будет доступен для соединения:

BOOL WaitNamedPipe( LPCTSTR lpszPipeName, DWORD dwTimeout);

LPCTSTR lpszPipeName – имя канала;

DWORD dwTimeout – время ожидания (в миллисекундах). В качестве параметра могут быть переданы константы:

NMPWAIT\_WAIT\_FOREVER - ожидание выполняется бесконечно долго, NMPWAIT\_USE\_DEFAULT\_WAIT - ожидание выполняется в течении периода времени, указанного при вызове функции CreateNamedPipe.

Возвращаемое значение: если канал доступен до истечения периода времени, то

функция возвращает TRUE, иначе – FALSE.

Общая последовательность вызовов со стороны сервера:

hNp = CreateNamedPipe("\\\\.\\pipe\\my\_pipe", …);

while (… /\* Цикл продолжается вплоть до завершения работы сервера.\*/)

{

ConnectNamedPipe(hNp, NULL); //подключаем клиента

while (ReadFile(hNp, Request, …) //пока он не закрыл соединение на своей стороне принимаем сообщения

{

…

WriteFile(hNp, Response, …); //шлем клиенту ответ

}

DisconnectNamedPipe(hNp); //отсоединяем клиента

}

CloseHandle(hNp);

Общая последовательность вызовов со стороны клиента:

WaitNamedPipe("\\\\ServerName\\pipe\\my\_pipe", NMPWAIT\_WAIT\_FOREVER);

//ждем разрешение сервера на соединение

hNp = CreateFile("\\\\ServerName\\pipe\\my\_pipe", …); //подключаемся к каналу

while (…/\*Цикл выполняется до тех пор, пока не прекратятся запросы.\*/

{

WriteFile(hNp, Request, …); //шлем запрос

…

ReadFile(hNp, Response); //получаем ответ

}

CloseHandle (hNp); /\* Разорвать соединение с сервером. \*/

**2.1. Реализовать между одним клиентом и сервером обмен данными, вводимыми с консоли на стороне клиента и возвращаемыми сервером обратно до получения команды exit.**

Программа-сервер (файл Master) создает именованный канал для двунаправленного использования и ожидает подключения программы-клиента (slave).

Проверяем, корректно ли произошло подключение, затем входим в цикл получения команд от "клиента" с последующими эхо-ответами. При появлении команды exit со стороны клиента, сервер завершает работу, закрывает канал.

Клиент на своей стороне открывает канал, пишет в него и читает эхо-ответ. При вводе exit программа завершается.

**Исходный код сервера:**

#include <windows.h>

#include <stdio.h>

#include <conio.h>

int main()

{

// Флаг успешного создания канала

BOOL fConnected;

// Идентификатор канала Pipe

HANDLE hNamedPipe;

// Имя создаваемого канала Pipe

LPCSTR lpszPipeName = "\\\\.\\pipe\\$MyPipe$";

// Буфер для передачи данных через канал

char szBuf[512];

// Количество байт данных, принятых через канал

DWORD cbRead;

// Количество байт данных, переданных через канал

DWORD cbWritten;

printf("Named pipe was creating \n");

// Создаем канал Pipe, имеющий имя lpszPipeName

hNamedPipe = CreateNamedPipe(lpszPipeName, //имя канала

PIPE\_ACCESS\_DUPLEX, // режим котрытия канала - двунаправленный

//параметры режима pipe:

PIPE\_TYPE\_MESSAGE //Данные записываются в канал в виде потока сообщений

| PIPE\_READMODE\_MESSAGE //Данные считываются в виде потока сообщений

| PIPE\_WAIT,// блокирующий режим

PIPE\_UNLIMITED\_INSTANCES, //неограниченное кол-во "подключений" к каналу

512, 512, //кол-во байт входного и вызодного буферов

0, //тайм-аут в 50 миллисекунд (по умолчанию)

NULL ); // дескриптор безопасности по умолчанию

// Если возникла ошибка, выводим ее код и

// завершаем работу приложения

if(hNamedPipe == INVALID\_HANDLE\_VALUE)

{

fprintf(stdout,"CreateNamedPipe: Error %ld\n",

GetLastError());

getch();

return 0;

}

// Выводим сообщение о начале процесса создания канала

fprintf(stdout,"Waiting for connect...\n");

// Ожидаем соединения со стороны клиента

fConnected = ConnectNamedPipe(hNamedPipe, // имя канала

NULL); // overlapped=null

// При возникновении ошибки выводим ее код

if(!fConnected)

{

switch(GetLastError())

{

case ERROR\_NO\_DATA:

fprintf(stdout,"ConnectNamedPipe: ERROR\_NO\_DATA");

getch();

CloseHandle(hNamedPipe);

return 0;

break;

case ERROR\_PIPE\_CONNECTED:

fprintf(stdout,

"ConnectNamedPipe: ERROR\_PIPE\_CONNECTED");

getch();

CloseHandle(hNamedPipe);

return 0;

break;

case ERROR\_PIPE\_LISTENING:

fprintf(stdout,

"ConnectNamedPipe: ERROR\_PIPE\_LISTENING");

getch();

CloseHandle(hNamedPipe);

return 0;

break;

case ERROR\_CALL\_NOT\_IMPLEMENTED:

fprintf(stdout,

"ConnectNamedPipe: ERROR\_CALL\_NOT\_IMPLEMENTED");

getch();

CloseHandle(hNamedPipe);

return 0;

break;

default:

fprintf(stdout,"ConnectNamedPipe: Error %ld\n",

GetLastError());

getch();

CloseHandle(hNamedPipe);

return 0;

break;

}

CloseHandle(hNamedPipe);

getch();

return 0;

} // Выводим сообщение об успешном создании канала

fprintf(stdout,"\nConnected. Waiting for command...\n");

// Цикл получения команд из канала

while(1)

{ if(ReadFile(hNamedPipe, szBuf, 512, &cbRead, NULL))

{

// Выводим принятую команду на консоль

printf("Received: <%s>\n", szBuf);

// Если пришла команда "exit",

// завершаем работу приложения

if(!strcmp(szBuf, "exit"))

break;

// отправляем эхо-ответ

if(!WriteFile(hNamedPipe, szBuf, strlen(szBuf) + 1, &cbWritten, NULL)) break;

}

else

{getch();

break; }

}

CloseHandle(hNamedPipe);

return 0;}

**Исходный код клиента:**

#include <windows.h

#include <stdio.h>

#include <conio.h>

int main(int argc, char \*argv[])

{ // Идентификатор канала Pipe

HANDLE hNamedPipe;

// Количество байт, переданных через канал

DWORD cbWritten;

// Количество байт, принятых через канал

DWORD cbRead;

// Буфер для передачи данных

char szBuf[256];

// Буфер для имени канала Pipe

LPCSTR szPipeName="\\\\.\\pipe\\$MyPipe$";

printf("Named pipe client demo\n");

printf("Syntax: pipec [servername]\n");

// открываем handle нашего именованого pipe

hNamedPipe = CreateFile(

szPipeName, // pipe name

GENERIC\_READ | // read and write access

GENERIC\_WRITE,

0, // no sharing

NULL, // default security attributes

OPEN\_EXISTING, // opens existing pipe

0, // default attributes

NULL); // no template file

// Если возникла ошибка, выводим ее код и

// завершаем работу приложения

if(hNamedPipe == INVALID\_HANDLE\_VALUE)

{

fprintf(stdout,"CreateFile: Error %ld\n",

GetLastError());

getch();

return 0;

}

// Выводим сообщение о создании канала

fprintf(stdout,"\nConnected. Type 'exit' to terminate\n");

// Цикл обмена данными с серверным процессом

while(1)

{ // Выводим приглашение для ввода команды

printf("cmd>");

// Вводим текстовую строку

gets(szBuf);

// Передаем введенную строку серверному процессу

// в качестве команды

if(!WriteFile(hNamedPipe,szBuf,strlen(szBuf)+1,&cbWritten, NULL)) break;

// Получаем эту же команду обратно от сервера

if(ReadFile(hNamedPipe, szBuf, 512, &cbRead, NULL))

printf("Received back: <%s>\n", szBuf);

// В ответ на команду "exit" завершаем цикл

// обмена данными с серверным процессом

if(!strcmp(szBuf, "exit"))

break;

}

// Закрываем идентификатор канала

CloseHandle(hNamedPipe);

return 0;

}

**Результат выполнения программы:**

Сервер:

C:\lab\_6\1\_3>master.exe

Named pipe was creating

Waiting for connect...

Connected. Waiting for command...

Received: <Hello>

Received: <I'm here>

Received: <exit>

Клиент:

C:\lab\_6\1\_3>slave.exe

Named pipe client demo

Syntax: pipec [servername]

Connected. Type 'exit' to terminate

cmd>Hello

Received back: <Hello>

cmd>I'm here

Received back: <I'm here>

cmd>exit

Запускаем в одном окне сервер. Он выводит сообщения о том, что канал создан и ожидает подключения клиента. Запускаем в другом окне клиента.

После запуска клиента на стороне сервера мы получаем уведомление о том, что создано новое подключение, теперь мы можем получать сообщения от клиента.

Клиент после запуска выводит уведомления о том, что он подключился к серверу и ожидает ввода команд в строке cmd> . Вводимые в командной строке символы он пересылает серверу и сразу выводит возвращенный эхо-ответ от него. Для завершения сеанса обмена следует ввести зарезервированную команду exit на стороне клиента, после ее доставки серверу, он завершает работу.

Сервер работает с одним клиентом, поэтому ему не нужно вызывать функции отсоединения клиента (по завершению сервера, клиент тоже завершается).

**2.2. Реализовать между сервером и множеством клиентов обмен данными, вводимыми с консоли на стороне клиента и возвращаемыми сервером обратно до получения команды exit.**

Сервер, как и ранее, создает все необходимые ресурсы и переходит в состояние ожидания соединений. Именованный канал создается для чтения и записи. Передача происходит сообщениями, функции передачи и приема блокируются до их окончания. Клиент после соединения с сервером начинает чтение сообщений с консоли, пока не встретит слово «exit». По данному слову и клиент, и сервер завершают свою работу.

**Исходный код сервера:**

#include <windows.h>

#include <stdio.h>

#define SIZE\_OF\_BUF 1000

DWORD WINAPI threadHandler(LPVOID param){

HANDLE hNamedPipe = (HANDLE)param;

DWORD readbytes, writebytes; //число байт прочитанных и переданных

char buf[SIZE\_OF\_BUF]; // Буфер для передачи данных через канал

while (1)

{

// Получаем очередную команду через канал Pipe

if (ReadFile(hNamedPipe, buf, SIZE\_OF\_BUF, &readbytes, NULL))

{

// Посылаем эту команду обратно клиентскому

// приложению

if (!WriteFile(hNamedPipe, buf, strlen(buf) + 1, &writebytes, NULL))

break;

// Выводим принятую команду на консоль

printf("Get client msg: %s\n", buf);

// Если пришла команда "exit",

// завершаем работу приложения

if (!strncmp(buf, "exit", 4))

break;

}

else

{

fprintf(stdout, "ReadFile: Error %ld\n", GetLastError());

getchar();

break;

}

}

CloseHandle(hNamedPipe);

ExitThread(0);

}

int main()

{

HANDLE hNamedPipe; // Идентификатор канала Pipe

HANDLE t; //Для потока

LPCSTR lpszPipeName = "\\\\.\\pipe\\$$MyPipe$$"; // Имя создаваемого канала Pipe

printf("Server is started. Try to create named pipe\n");

// Создаем канал Pipe, имеющий имя lpszPipeName

while (1){

hNamedPipe = CreateNamedPipe(

lpszPipeName, //имя канала

PIPE\_ACCESS\_DUPLEX, //доступ и на чтение и на запись

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, //передача сообщений (как чтение,

//так и запись)

5, //максимальное число экземпляров каналов равно 5 (число клиентов)

SIZE\_OF\_BUF, SIZE\_OF\_BUF, 5000, NULL); //размеры выходного и входного буферов канала, 5

//секунд - длительность для функции WaitNamedPipe

if (hNamedPipe == INVALID\_HANDLE\_VALUE) // Если возникла ошибка, выводим ее код и завершаем

//работу приложения

{

fprintf(stdout, "CreateNamedPipe: Error %ld\n", GetLastError());

getchar();

return 1000;

}

printf("Named pipe created successfully\n");

// Выводим сообщение о начале процесса создания канала

printf("waiting for connect\n");

// Ожидаем соединения со стороны клиента

if (!ConnectNamedPipe(hNamedPipe, NULL))

{

// При возникновении ошибки выводим ее код

printf("error with function ConnectNamedPipe\n");

getchar();

CloseHandle(hNamedPipe);

return 1001;

}

// Выводим сообщение об успешном создании канала

fprintf(stdout, "Client connected\n");

t = CreateThread(NULL, 0, threadHandler, (LPVOID)hNamedPipe, 0, NULL);

CloseHandle(t);

}

printf("server is ending\n press any key\n");

getchar();

return 0;

}

**Исходный код клиента:**

#include <windows.h>

#include <stdio.h>

#define SIZE\_OF\_BUF 1000

int main(int argc, char \*argv[]) {

HANDLE hNamedPipe; // Идентификатор канала Pipe

DWORD readbytes,writebytes; // количество байт принятых и переданных в канал

char buf[SIZE\_OF\_BUF]; // Буфер для передачи данных

printf("Client is started\n Try to use WaitNamedPipe\n"); //"ожидаем" пока освободится канал

LPCSTR szPipeName = "\\\\.\\pipe\\$$MyPipe$$";

if (!WaitNamedPipe(szPipeName, NMPWAIT\_WAIT\_FOREVER)) {

printf("pipe wasn't created\n getlasterror = %d", GetLastError());

getchar();

return 1000;

} // Создаем канал с процессом-сервером

hNamedPipe = CreateFile(szPipeName, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL); // Если возникла ошибка, выводим ее код и

// завершаем работу приложения

if(hNamedPipe == INVALID\_HANDLE\_VALUE) {

fprintf(stdout,"CreateFile: Error %ld\n",GetLastError());

getchar();

return 1001;

} // Выводим сообщение о создании канала

printf("successfully connected\n input message\n"); // Цикл обмена данными с серверным процессом

while(1) { // Выводим приглашение для ввода команды printf("cmd>"); // Вводим текстовую строку

fgets(buf,SIZE\_OF\_BUF,stdin); // Передаем введенную строку серверному процессу // в качестве команды

if(!WriteFile(hNamedPipe, buf, strlen(buf) + 1, &writebytes, NULL)) {

printf("connection refused\n");

break;

} // Получаем эту же команду обратно от сервера

if(ReadFile(hNamedPipe, buf, SIZE\_OF\_BUF, &readbytes, NULL))

printf("Received from server: %s\n", buf); // Если произошла ошибка, выводим ее код и // завершаем работу приложения

else {

fprintf(stdout,"ReadFile: Error %ld\n", GetLastError());

getchar();

break;

} // В ответ на команду "exit" завершаем цикл // обмена данными с серверным процессом

if(!strncmp(buf, "exit",4))

break;

} // Закрываем идентификатор канала

//CloseHandle(hNamedPipe);

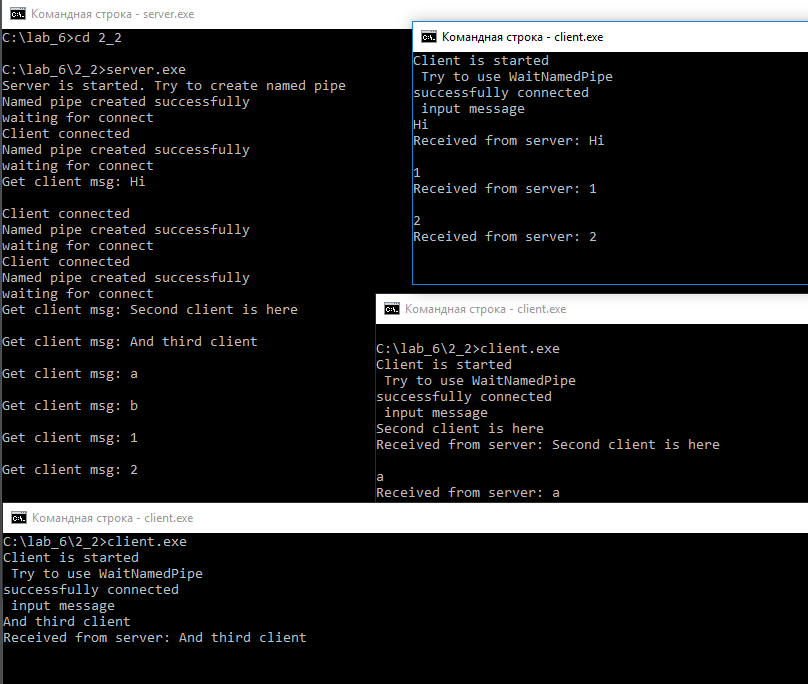
printf("client is ending\n Press any key\n");

getchar();

return 0;

}

**Результат выполнения программы:**

****Рис.3. Результат выполнения программы

После запуска сервера было запущено 3 клиента в других терминалах. Клиенты по очереди передавали сообщения серверу, а тот отправлял им эти сообщения назад. Каждый клиент работал со своим экземпляром канала, поэтому передаваемые данные не перемешивались и сервер правильно их обрабатывал.

**2.3. Модифицировать приложение из пункта 2.2 для сетевого обмена информацией.**

Для совместной работы компьютеры нужно подсоединить к одной домашней группе. Так же необходимо установить поле DACL (Discretionary Access Control List) защиты объекта в NULL (разрешение всем пользователям и группам доступа к объекту). Параметры защиты именованного канала задаются с помощью структуры SECURITY\_ATTRIBUTES, которая указывается последним параметром в функции CreateNamedPipe.

**Исходный код сервера:**

#include <windows.h>

#include <stdio.h>

#define SIZE\_OF\_BUF 1000

int main()

{

HANDLE hNamedPipe; // Идентификатор канала Pipe

// Имя создаваемого канала Pipe

LPSTR lpszPipeName = "\\\\.\\pipe\\$$MyPipe$$";

char buf[SIZE\_OF\_BUF]; // Буфер для передачи данных через канал

DWORD readbytes, writebytes; //число байт прочитанных и переданных

printf("Server is started. Try to create named pipe\n");

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Создание SECURITY\_ATTRIBUTES и SECURITY\_DESCRIPTOR объектов

SECURITY\_ATTRIBUTES sa;

SECURITY\_DESCRIPTOR sd;

// Инициализация SECURITY\_DESCRIPTOR значениями по-умолчанию

if (InitializeSecurityDescriptor(&sd,

SECURITY\_DESCRIPTOR\_REVISION) == 0)

{

printf("InitializeSecurityDescriptor failed with error %d\n",

GetLastError());

return 50000;

}

// Установка поля DACL в SECURITY\_DESCRIPTOR в NULL

if (SetSecurityDescriptorDacl(&sd, TRUE, NULL, FALSE) == 0)

{

printf("SetSecurityDescriptorDacl failed with error %d\n",

GetLastError());

return 50001;

}

// Установка SECURITY\_DESCRIPTOR в структуре SECURITY\_ATTRIBUTES

sa.nLength = sizeof(SECURITY\_ATTRIBUTES);

sa.lpSecurityDescriptor = &sd;

sa.bInheritHandle = FALSE; //запрещение наследования

hNamedPipe = CreateNamedPipe(

lpszPipeName, //имя канала

PIPE\_ACCESS\_DUPLEX, //доступ и на чтение и на запись //передача сообщений (как чтение, так и запись)

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, 5, //максимальное число экземпляров каналов равно 5 (числоклиентов)

SIZE\_OF\_BUF, SIZE\_OF\_BUF, 5000, &sa); //размеры выходного и входного буферов канала, 5 секунд - длительность для функции WaitNamedPipe

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (hNamedPipe == INVALID\_HANDLE\_VALUE) // Если возникла ошибка,

//выводим ее код и завершаем работу приложения

{

fprintf(stdout, "CreateNamedPipe: Error %ld\n", GetLastError());

getchar();

return 1000;

}

printf("Named pipe created successfully\n");

// Выводим сообщение о начале процесса создания канала

printf("waiting for connect\n");

// Ожидаем соединения со стороны клиента

if (!ConnectNamedPipe(hNamedPipe, NULL))

{

// При возникновении ошибки выводим ее код

printf("error with function ConnectNamedPipe\n");

getchar();

CloseHandle(hNamedPipe);

return 1001;

}

// Выводим сообщение об успешном создании канала

fprintf(stdout, "Client connected\n");

// Цикл получения команд через канал

while (1)

{

// Получаем очередную команду через канал Pipe

if (ReadFile(hNamedPipe, buf, SIZE\_OF\_BUF, &readbytes, NULL))

{

// Посылаем эту команду обратно клиентскому приложению

if (!WriteFile(hNamedPipe, buf, strlen(buf) + 1, &writebytes, NULL))

break;

// Выводим принятую команду на консоль

printf("Get client msg: %s\n", buf);

// Если пришла команда "exit",

// завершаем работу приложения

if (!strncmp(buf, "exit", 4))

break;

}

else

{

fprintf(stdout, "ReadFile: Error %ld\n", GetLastError());

getchar();

break;

}

}

CloseHandle(hNamedPipe);

printf("server is ending\n press any key\n");

getchar();

return 0;

}

**Исходный код клиента:**

#include <windows.h>

#include <stdio.h>

#define SIZE\_OF\_BUF 1000

int main(int argc, char \*argv[])

{

HANDLE hNamedPipe; // Идентификатор канала Pipe

DWORD readbytes, writebytes; // количество байт принятых и

//переданных в канал

char buf[SIZE\_OF\_BUF]; // Буфер для передачи данных

printf("Client is started\n Try to use WaitNamedPipe\n");

//"ожидаем" пока освободится канал

char \*pipename = "\\\\192.168.0.5\\pipe\\MyPipe"; //адрес сервера и имя канала "ожидаем" пока освободится канал

if (!WaitNamedPipe(pipename, NMPWAIT\_WAIT\_FOREVER))

{

printf("pipe wasn't created\n getlasterror = %d",

GetLastError());

getchar();

return 1000;

}

// Создаем канал с процессом-сервером

hNamedPipe = CreateFile("\\\\.\\pipe\\$$MyPipe$$", GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

// Если возникла ошибка, выводим ее код и

// завершаем работу приложения

if (hNamedPipe == INVALID\_HANDLE\_VALUE)

{

fprintf(stdout, "CreateFile: Error %ld\n", GetLastError());

getchar();

return 1001;

}

// Выводим сообщение о создании канала

printf("successfully connected\n input message\n");

// Цикл обмена данными с серверным процессом

while (1)

{

// Выводим приглашение для ввода команды

printf("cmd>");

// Вводим текстовую строку

fgets(buf, SIZE\_OF\_BUF, stdin);

// Передаем введенную строку серверному процессу

// в качестве команды

if (!WriteFile(hNamedPipe, buf, strlen(buf) + 1, &writebytes,

NULL))

{

printf("connection refused\n");

break;

}

// Получаем эту же команду обратно от сервера

if (ReadFile(hNamedPipe, buf, SIZE\_OF\_BUF, &readbytes, NULL))

printf("Received from server: %s\n", buf);

// Если произошла ошибка, выводим ее код и

// завершаем работу приложения

else

{

fprintf(stdout, "ReadFile: Error %ld\n", GetLastError());

getchar();

break;

}

// В ответ на команду "exit" завершаем цикл

// обмена данными с серверным процессом

if (!strncmp(buf, "exit", 4))

break;

}

// Закрываем идентификатор канала

CloseHandle(hNamedPipe);

printf("client is ending\n Press any key\n");

getchar();

return 0;

}

В связи с тем, что я имею доступ только к одному компьютеру, данную программу не удалось протестировать.

**3. Сокеты:**

Сам по себе сокет – это оконечная точка соединения, которая идентифицируется 4 значениями: IP адрес отправителя, порт отправителя, IP адрес получателя, порт получателя.

Порт – идентификатор процесса в ОС с точки зрения сетевого взаимодействия. Порт напрямую связан с протоколом. Например, в ОС могут быть два процесса с одинаковым номером порта, но использующие при этом разные протоколы. В качестве номера порта желательно выбирать значение большее 1024 (т.к. это зарезервированные порты).

**3.1. Разработать программу локального обмена сокетами с использованием потокового протокола с установлением соединения (TCP в стеке TCP/IP).**

Для потоковых протоколов (к которым относится протокол TCP в стеке TCP/IP) необходимо применять средства, позволяющие определить границы сообщений в передаваемых данных, так как данный вид протоколов имеет дело с доставкой только потока байт (при этом гарантируется порядок доставки). В качестве функций, обеспечивающих передачу по разделителям, используются:

• int recvn(SOCKET fd, char \*bp, size\_t len); - функция, которая читает из сокета заданное число байт;

• int sendn(SOCKET s, char\* buf, int lenbuf, int flags); - функция, которая передает в сокет заданное число байт;

• int sendLine(int sock, char\* str); - функция, которая добавляет к исходным данным символ-разделитель сообщений и передает сообщение функции sendn;

• int recvLine(SOCKET s, char\* buffer, int buffSize); - функция, которая читает буфер сокета до разделителя и полученную строку возвращает (разделитель исключается).

**Исходный код сервера:**

#include <stdlib.h>

#include <stdio.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#define MSG\_WAITALL 0x8

#define MAX\_STR\_LEN 255

#define SIZE\_OF\_BUF 255

int recvn(SOCKET fd, char \*bp, size\_t len) {

return recv(fd, bp, len, MSG\_WAITALL);

}

int sendn(SOCKET s, char\* buf, int lenbuf, int flags) {

int bytesSended = 0; //

int n; //

while (bytesSended < lenbuf) {

n = send(s, buf + bytesSended, lenbuf - bytesSended, flags);

if (n < 0) {

printf("Error with send in sendn\n");

break;

}

bytesSended += n;

}

return (n == -1 ? -1 : bytesSended);

}

int recvLine(SOCKET sock, char\* buffer, int buffSize) { //функция приема сообщения

char\* buff = buffer; //указатель на начало внешнего буфера

char\* currPosPointer; //указатель для работы со временным буфером

int count = 0; //число прочитанных символов (без удаления из буфера сокета)

char tempBuf[100]; //временный буфер для приема

char currChar; //текущий анализируемый символ (ищем разделитель)

int tmpcount = 0;

while (--buffSize > 0){

if (--count <= 0) {

recvn(sock, tempBuf, tmpcount);

count = recv(sock, tempBuf, sizeof (tempBuf), MSG\_PEEK);

if (count <= 0) { return count; }

currPosPointer = tempBuf;

tmpcount = count;

}

currChar = \*currPosPointer++;

\*buffer++ = currChar;

if (currChar == '\n') {

\*(buffer - 1) = '\0';

recvn(sock, tempBuf, tmpcount - count + 1);

return buffer - buff-1;

}

}

return -1;

}

int sendLine(int sock, char\* str) {

char tempBuf[MAX\_STR\_LEN];

strcpy(tempBuf, str);

if(tempBuf[strlen(tempBuf)-1]!='\n')

strcat(tempBuf, "\n");

return sendn(sock, tempBuf, strlen(tempBuf), 0);

}

DWORD WINAPI threadHandler(LPVOID param){

SOCKET client\_socket = (SOCKET)param;

if (client\_socket == INVALID\_SOCKET) {

printf("error with accept socket. GetLasterror= %ld\n", GetLastError());

return 1003;

}

char buf[SIZE\_OF\_BUF]; //буфер приема и передачи сообщения

int readbytes; //число прочитанных байт

while (1) {

if ((readbytes = recvLine(client\_socket, buf, SIZE\_OF\_BUF)) == 0) {

printf("Connection refused\n");

break;

}

else if (readbytes == -1) {

printf("buf is small\n");

return 2000;

}

printf("get msg from client \"%s\" with size= %d\n", buf, readbytes);

sendLine(client\_socket, buf); //sendn(client\_socket,buf,readbytes,0); //шлем сообщение обратно клиенту

if (strncmp(buf, "exit", 4) == 0) break;

}

closesocket(client\_socket);

return 0;

}

int main(void) {

//используется для инициализации библиотеки сокетов WS

WSADATA WSStartData;

if (WSAStartup(MAKEWORD(2, 0), &WSStartData) != 0) {

printf("WSAStartup failed with error: %ld\n", GetLastError());

return 100;

} //создание сокета

SOCKET server\_socket; //по умолчанию используется протокол tcp

printf("Server is started.\nTry to create socket -----------------");

if((server\_socket = socket( AF\_INET, SOCK\_STREAM, 0 )) ==INVALID\_SOCKET) {

printf("error with creation socket. GetLasterror= %ld\n",GetLastError());

return 1000;

}

printf("CHECK\n"); //Привязывание сокета конкретному IP и номеру порта

struct sockaddr\_in sin; sin.sin\_addr.s\_addr=inet\_addr("127.0.0.1"); // используем локальную машину

sin.sin\_port=htons(7500); // может быть любым кроме зарезервированных

sin.sin\_family=AF\_INET; printf("Try to bind socket -------------------");

if ( bind( server\_socket, (struct sockaddr \*)&sin, sizeof(sin) ) !=0 ) {

printf("error with bind socket. GetLasterror= %ld\n",GetLastError());

return 1001;

}

printf("CHECK\n"); //делаем сокет прослушиваемым

printf("Try to set socket listening ----------");

if(listen(server\_socket,5 )!=0) {

printf("error with listen socket. GetLasterror= %ld\n",GetLastError());

return 1002;

}

printf("CHECK\n");

printf("Server starts listening\n"); //Ждем клиента. Создаем пустую структуру, которая будет содержать параметры сокета, инициирующего соединение

struct sockaddr\_in from;

int fromlen=sizeof(from); // начинаем "слушать" входящие запросы на подключение

SOCKET client\_socket=accept(server\_socket,(struct sockaddr\*)&from,&fromlen);

if(client\_socket==INVALID\_SOCKET) {

printf("error with accept socket. GetLasterror= %ld\n",GetLastError());

return 1003;

}

printf("get client with IP= %s, port = %d\n",inet\_ntoa(from.sin\_addr),ntohs(from.sin\_port));

char buf[SIZE\_OF\_BUF]; //буфер приема и передачи сообщения

int readbytes; //число прочитанных байт

while(1) {

if((readbytes=recvLine(client\_socket,buf,SIZE\_OF\_BUF))==0) {

printf("Connection refused\n");

break;

}

else if(readbytes==-1) {

printf("buf is small\n");

return 2000;

}

printf("get msg from client \"%s\" with size= %d\n",buf,readbytes);

sendLine(client\_socket,buf); //sendn(client\_socket,buf,readbytes,0); //шлем сообщение обратно клиенту

if (strncmp(buf, "exit", 4) == 0) break;

}

closesocket(client\_socket);

closesocket(server\_socket);

return 0;

}

**Исходный код клиента:**

#include <stdlib.h>

#include <stdio.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#define MAX\_STR\_LEN 255

#define SIZE\_OF\_BUF 255

#define MSG\_WAITALL 0x8

int recvn(SOCKET fd, char \*bp, size\_t len) {

return recv(fd, bp, len, MSG\_WAITALL);

}

int sendn(SOCKET s, char\* buf, int lenbuf, int flags) {

int bytesSended = 0; //

int n; //

while (bytesSended < lenbuf) {

n = send(s, buf + bytesSended, lenbuf - bytesSended, flags);

if (n < 0) {

printf("Error with send in sendn\n");

break;

}

bytesSended += n;

}

return (n == -1 ? -1 : bytesSended);

}

int recvLine(SOCKET sock, char\* buffer, int buffSize) { //функция приема сообщения

char\* buff = buffer; //указатель на начало внешнего буфера

char\* currPosPointer; //указатель для работы со временным буфером

int count = 0; //число прочитанных символов (без удаления из буфера сокета)

char tempBuf[100]; //временный буфер для приема

char currChar; //текущий анализируемый символ (ищем разделитель)

int tmpcount = 0;

while (--buffSize > 0){

if (--count <= 0) {

recvn(sock, tempBuf, tmpcount);

count = recv(sock, tempBuf, sizeof (tempBuf), MSG\_PEEK);

if (count <= 0) { return count; }

currPosPointer = tempBuf;

tmpcount = count;

}

currChar = \*currPosPointer++;

\*buffer++ = currChar;

if (currChar == '\n') {

\*(buffer - 1) = '\0';

recvn(sock, tempBuf, tmpcount - count + 1);

return buffer - buff-1;

}

}

return -1;

}

int sendLine(int sock, char\* str) {

char tempBuf[MAX\_STR\_LEN];

strcpy(tempBuf, str);

if(tempBuf[strlen(tempBuf)-1]!='\n')

strcat(tempBuf, "\n");

return sendn(sock, tempBuf, strlen(tempBuf), 0);

}

int main(void) { //используется для инициализации библиотеки сокетов

WSADATA WSStartData; //Инициализация WinSock и проверка его запуска

if (WSAStartup(MAKEWORD(2, 0), &WSStartData) != 0) {

printf("WSAStartup failed with error: %ld\n", GetLastError());

return 100;

}

int er\_code=0; // инициализация клиентского сокета

printf("Client is started.\nTry to create socket\n");

int client\_socket = socket( AF\_INET, SOCK\_STREAM, 0 );

printf("socket created successfully\n");

struct sockaddr\_in sin;

sin.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

sin.sin\_port=htons(7500);

sin.sin\_family=AF\_INET; // установливаем TCP-соединение

printf("try to connect to server\n");

if(connect(client\_socket, (struct sockaddr \*) &sin,sizeof(sin))!=0) {

printf("connect failed with error: %d\n", er\_code);

return SOCKET\_ERROR;

}

printf("Client connected sucessfully\nEnter msg to send\n---------------------\n");

char buf[SIZE\_OF\_BUF]; //буфер для приема и передачи сообщений

while(1) {

fgets(buf,SIZE\_OF\_BUF,stdin);

printf("client sended msg: %s",buf);

sendLine(client\_socket,buf);

recvLine(client\_socket,buf,SIZE\_OF\_BUF);

printf("get msg from serv: \"%s\"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",buf);

} // заканчиваем работу с сокетом клиента

closesocket(client\_socket);

return 0;

}

**Результат выполнения программы:**

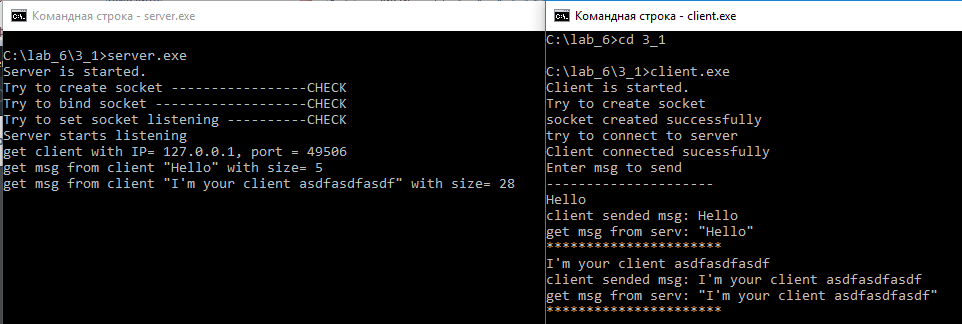


Рис.4. Результат обмена с использованием сокетов

**3.2. Модифицировать программу для локального обмена с множеством клиентов и с доступом к общему ресурсу.**

**Исходный код модифицированного сервера:**

#include <stdlib.h>

#include <stdio.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#define MAX\_STR\_LEN 255

#define SIZE\_OF\_BUF 255

#define MSG\_WAITALL 0x8

int recvn(SOCKET fd, char \*bp, size\_t len) {

return recv(fd, bp, len, MSG\_WAITALL);

}

int sendn(SOCKET s, char\* buf, int lenbuf, int flags) {

int bytesSended = 0; //

int n; //

while (bytesSended < lenbuf) {

n = send(s, buf + bytesSended, lenbuf - bytesSended, flags);

if (n < 0) {

printf("Error with send in sendn\n");

break;

}

bytesSended += n;

}

return (n == -1 ? -1 : bytesSended);

}

int recvLine(SOCKET sock, char\* buffer, int buffSize) { //функция приема сообщения

char\* buff = buffer; //указатель на начало внешнего буфера

char\* currPosPointer; //указатель для работы со временным буфером

int count = 0; //число прочитанных символов (без удаления из буфера сокета)

char tempBuf[100]; //временный буфер для приема

char currChar; //текущий анализируемый символ (ищем разделитель)

int tmpcount = 0;

while (--buffSize > 0){

if (--count <= 0) {

recvn(sock, tempBuf, tmpcount);

count = recv(sock, tempBuf, sizeof (tempBuf), MSG\_PEEK);

if (count <= 0) { return count; }

currPosPointer = tempBuf;

tmpcount = count;

}

currChar = \*currPosPointer++;

\*buffer++ = currChar;

if (currChar == '\n') {

\*(buffer - 1) = '\0';

recvn(sock, tempBuf, tmpcount - count + 1);

return buffer - buff-1;

}

}

return -1;

}

int sendLine(int sock, char\* str) {

char tempBuf[MAX\_STR\_LEN];

strcpy(tempBuf, str);

if(tempBuf[strlen(tempBuf)-1]!='\n')

strcat(tempBuf, "\n");

return sendn(sock, tempBuf, strlen(tempBuf), 0);

}

DWORD WINAPI threadHandler(LPVOID param){

SOCKET client\_socket = (SOCKET)param;

if (client\_socket == INVALID\_SOCKET) {

printf("error with accept socket. GetLasterror= %ld\n", GetLastError());

return 1003;

}

char buf[SIZE\_OF\_BUF]; //буфер приема и передачи сообщения

int readbytes; //число прочитанных байт

while (1) {

if ((readbytes = recvLine(client\_socket, buf, SIZE\_OF\_BUF)) == 0) {

printf("Connection refused\n");

break;

}

else if (readbytes == -1) {

printf("buf is small\n");

return 2000;

}

printf("get msg from client \"%s\" with size= %d\n", buf, readbytes);

sendLine(client\_socket, buf); //sendn(client\_socket,buf,readbytes,0); //шлем сообщение обратно клиенту

if (strncmp(buf, "exit", 4) == 0) break;

}

closesocket(client\_socket);

return 0;

}

int main(void) {

//используется для инициализации библиотеки сокетов

WSADATA WSStartData; //Инициализация WinSock и проверка его запуска

if (WSAStartup(MAKEWORD(2, 0), &WSStartData) != 0) {

printf("WSAStartup failed with error: %ld\n", GetLastError());

return 100;

} //создание сокета

SOCKET server\_socket; //по умолчанию используется протокол tcp

printf("Server is started.\nTry to create socket -----------------");

if((server\_socket = socket( AF\_INET, SOCK\_STREAM, 0 )) ==INVALID\_SOCKET) {

printf("error with creation socket. GetLasterror= %ld\n",GetLastError());

return 1000;

}

printf("CHECK\n"); //Привязывание сокета конкретному IP и номеру порта

struct sockaddr\_in sin;

sin.sin\_addr.s\_addr=INADDR\_ANY; // используем все интерфейсы

sin.sin\_port=htons(7500); // может быть любым кроме зарезервированных

sin.sin\_family=AF\_INET; printf("Try to bind socket -------------------");

if ( bind( server\_socket, (struct sockaddr \*)&sin, sizeof(sin) ) !=0 ) {

printf("error with bind socket. GetLasterror= %ld\n",GetLastError());

return 1001;

}

printf("CHECK\n"); //делаем сокет прослушиваемым

printf("Try to set socket listening ----------");

if(listen(server\_socket,5 )!=0) {

printf("error with listen socket. GetLasterror= %ld\n",GetLastError());

return 1002;

}

printf("CHECK\n");

printf("Server starts listening\n"); //Ждем клиента. Создаем пустую структуру, которая будет содержать параметры сокета, инициирующего соединение

struct sockaddr\_in from;

int fromlen=sizeof(from); // начинаем "слушать" входящие запросы на подключение

while (SOCKET client\_socket = accept(server\_socket, (struct sockaddr\*)&from, &fromlen)){

printf("get client with IP= %s, port = %d\n", inet\_ntoa(from.sin\_addr), ntohs(from.sin\_port));

HANDLE t;

t = CreateThread(NULL, 0, threadHandler, (LPVOID)client\_socket, 0, NULL);

}

closesocket(server\_socket);

return 0;

}

Исходный код программы клиента остался без изменений.

**Результат выполнения программы:**

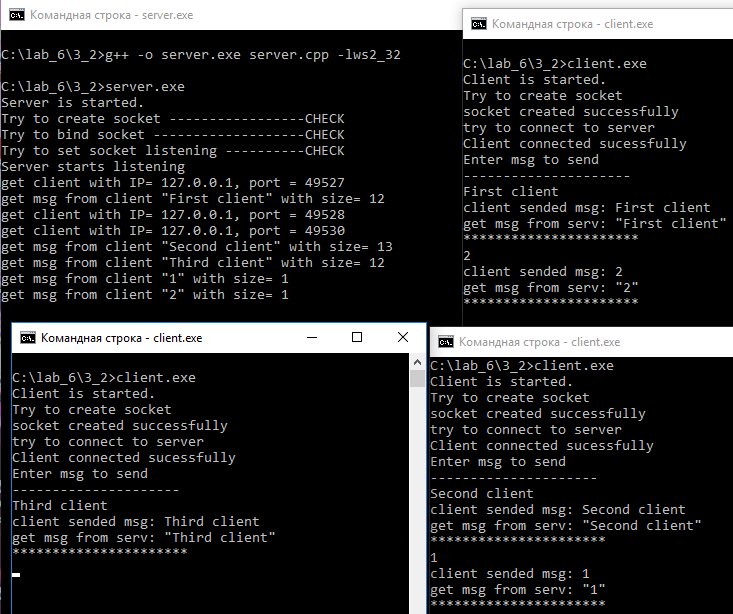


Рис.5. Взаимодействие сервера с множеством клиентов

**4. Сигналы в Windows**

Данное средство IPC в Windows не поддерживается. Однако, например, консольному приложению можно посылать сигналы CTRL+C и CTRL+BREAK. Система может посылать приложению сигналы: CTRL\_CLOSE\_EVENT, CTRL\_LOGOFF\_EVENT и CTRL\_SHUTDOWN\_EVENT, когда пользователь закрывает консоль, выходит из системы, или когда система завершается. По получению данных сигналов процесс может произвести корректное завершение.

С помощью функции SetConsoleCtrlHandler можно установить обработчик на данные сигналы, но отправить сигнал другому приложению мы не можем.

**4.1. Задание обработчика сигналов завершения для консольного приложения.**

Установка обработчика сигнала производится с помощью функции:

BOOL WINAPI SetConsoleCtrlHandler(PHANDLER\_ROUTINE HandlerRoutine, BOOL Add);

PHANDLER\_ROUTINE HandlerRoutine – функция-обработчик сигнала;

BOOL Add – флаг, значение TRUE которого говорит о добавлении обработчика в список, FALSE –удаление.

Возвращаемое значение: в случае успешной установки обработчика функция возвращает ненулевое значение, иначе – 0.

Функция-обработчик имеет следующее определение:

BOOL WINAPI HandlerRoutine(DWORD dwCtrlType);

DWORD dwCtrlType – тип сигнала, который был передан функции. Может принимать значение:

CTRL\_C\_EVENT - сигнал CTRL+C;

CTRL\_BREAK\_EVENT – сигнал CTRL+BREAK;

CTRL\_CLOSE\_EVENT – закрытие окна консоли (через меню либо через диспетчер

задач);

CTRL\_LOGOFF\_EVENT – пользователь выходит из системы;

CTRL\_SHUTDOWN\_EVENT – завершение работы системы.

Возвращаемое значение: если обработчик успешно обработал сигнал, он возвращает TRUE, иначе– FALSE.

Зарегистрированный обработчик должен проверять тип сигнала на возможность его обработки. Обработчики сигналов объединены в список. Когда приходит сигнал, вызывается последний зарегистрированный обработчик (при этом запускается отдельный поток). Если этот обработчик возвращает FALSE (он не обрабатывает этот сигнал), то вызывается следующий. Если все обработчики вернули FALSE, вызовется обработчик по-умолчанию, который по-умолчанию завершает процесс.

В качестве примера рассмотрим код из msdn. В нем происходит перехват сигналов CTRL+C, CTRL+BREAK. При этом обработчик смотрит, какой сигнал ему передан, и выводит его название. В качестве звуковой индикации работы приложение вызывает функцию Beep. Данная функция воспроизводит звуковой сигнал через динамик консоли с разной частотой и длительностью, задаваемыми ей через параметры.

В функции main регистрируется обработчик сигналов, затем главный поток работает в бесконечном цикле.

**Текст программы:**

#include <windows.h>

#include <stdio.h>

BOOL CtrlHandler(DWORD fdwCtrlType)

{

switch (fdwCtrlType) //тип сигнала

{

// Handle the CTRL-C signal.

case CTRL\_C\_EVENT:

printf("Ctrl-C event\n\n");

Beep(750, 300);

return(TRUE);

// CTRL-CLOSE: confirm that the user wants to exit.

case CTRL\_CLOSE\_EVENT:

Beep(600, 200);

printf("Ctrl-Close event\n\n");

return(TRUE);

// Pass other signals to the next handler.

case CTRL\_BREAK\_EVENT:

Beep(900, 200);

printf("Ctrl-Break event\n\n");

return FALSE;

case CTRL\_LOGOFF\_EVENT:

Beep(1000, 200);

printf("Ctrl-Logoff event\n\n");

return FALSE;

case CTRL\_SHUTDOWN\_EVENT:

Beep(750, 500);

printf("Ctrl-Shutdown event\n\n");

return FALSE;

default:

return FALSE;

}

}

int main(void)

{

if (SetConsoleCtrlHandler((PHANDLER\_ROUTINE)CtrlHandler, TRUE)){

printf("\nThe Control Handler is installed.\n");

printf("\n -- Now try pressing Ctrl+C or Ctrl+Break, or");

printf("\n try logging off or closing the console...\n");

printf("\n(...waiting in a loop for events...)\n\n");

while (1){}

}else{

printf("\nERROR: Could not set control handler");

return 1;

}

return 0;

}

**Результат выполнения программы:**

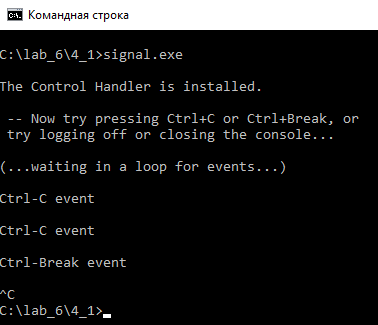


Рис.6. Результат работы программы обработчика сигналов завершения

Для сигналов воспроизводились разные по тональности звуки.

**4.2. Предложить собственную реализацию обработчика сигнала**

Сигналы в Windows не являются средством межпроцессного взаимодействия (в отличие от Unix), поэтому обработчики сигналов используются в основном для корректного завершения программы, для которой простой обрыв выполнения неприемлем, поэтому реализуем подтверждение пользователем выхода из программы.

**Текст программы:**

#include <windows.h>

#include <stdio.h>

BOOL CtrlHandler(DWORD fdwCtrlType)

{

switch (fdwCtrlType) //тип сигнала

{

// Handle the CTRL-C signal.

case CTRL\_C\_EVENT:

printf( "Ctrl-C event\n\n" );

Beep( 750, 300 );

return( TRUE );

//CTRL-CLOSE: confirm that the user wants to exit.

case CTRL\_CLOSE\_EVENT:

Beep(600, 200);

printf("Ctrl-Close event\n\n");

return(TRUE);

// Pass other signals to the next handler.

case CTRL\_BREAK\_EVENT:

Beep(900, 200);

printf("Ctrl-Break event\n\n");

printf("press ctrl+break one more time to leave\n");

SetConsoleCtrlHandler( (PHANDLER\_ROUTINE) CtrlHandler, FALSE );

return TRUE;

case CTRL\_LOGOFF\_EVENT:

Beep(1000, 200);

printf("Ctrl-Logoff event\n\n");

return FALSE;

case CTRL\_SHUTDOWN\_EVENT:

Beep(750, 500);

printf("Ctrl-Shutdown event\n\n");

return FALSE;

default:

return FALSE;

}

}

int main(void)

{

if (SetConsoleCtrlHandler((PHANDLER\_ROUTINE)CtrlHandler, TRUE)){

printf("\nThe Control Handler is installed.\n");

printf("\n -- Now try pressing Ctrl+C or Ctrl+Break, or");

printf("\n try logging off or closing the console...\n");

printf("\n(...waiting in a loop for events...)\n\n");

while (1){}

}else{

printf("\nERROR: Could not set control handler");

return 1;

}

return 0;

}

**Результат выполнения программы:**

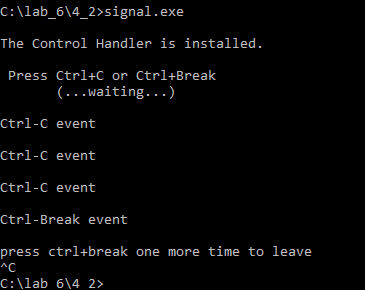


Рис.7. Результат работы программы обработчика сигналов завершения

**5. Разделяемая память**

Потоки одного процесса могут разделять общую память этого процесса. У каждого процесса – свое изолированное адресное пространство. Кроме рассмотренных выше средств передачи информации между процессами или потоками разных процессов, одно из наиболее эффективных – использование общей памяти, доступ к которой обеспечивается со стороны каждого процесса. ОС Windows поддерживает такое средство, как именованная, совместно используемая память.

Приведем системные функции, которые позволяют запрограммировать такое взаимодействие.

Первый участвующий в обмене информацией процесс создает объект "проекция файла" при помощи вызова функции CreateFileMapping() с параметром INVALID\_HANDLE\_VALUE и именем для объекта. Используя флажок PAGE\_READWRITE, задается доступ по чтению и записи в память через представление данных файла в адресном пространстве процесса. Процесс затем использует дескриптор объекта "проекция файла", возвращаемый функцией CreateFileMapping(), при вызове функции MapViewOfFile(). Эта функция создает представление файла в адресном пространстве процесса и возвращает указатель на представление данных файла для их дальнейшего использования.

Другой процесс может получить доступ к тем же данным при помощи вызова функции OpenFileMapping() с тем же самым именем, что и первый процесс, а затем использовать функцию MapViewOfFile(), чтобы получить свой указатель на представление данных файла.

Для записи данных в память используется функция CopyMemory(), первый аргумент которой – возвращаемый функцией MapViewOfFile() указатель, а следующие – характеризуют записываемые данные.

Когда процессу больше не нужен доступ к объекту "проекция файла в память", он должен вызвать функцию CloseHandle() для дальнейшего освобождения ресурса. При условии, что все дескрипторы закрыты (не осталось процессов, использующих этот ресурс), система может освободить секцию файла подкачки, используемого объектом.

**5.1. Создать программу, в которой первый процесс генерирует случайное число и записывает его в буфер, доступный второму процессу, откуда он его и считывает с последующим выводом.**

**Текст программы писателя:**

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#define BUF\_SIZE 256

TCHAR szName[] = TEXT("MyFileMappingObject");

TCHAR szMsg[] = TEXT("Message from first process");

HANDLE WINAPI mutex;

int main()

{

HANDLE hMapFile;

LPCTSTR pBuf;

mutex = CreateMutex(NULL, false, TEXT("SyncMutex"));

// create a memory, wicth two proccess will be working

hMapFile = CreateFileMapping(

INVALID\_HANDLE\_VALUE, // использование файла подкачки

NULL, // защита по умолчанию

PAGE\_READWRITE, // доступ к чтению/записи

0, // макс. размер объекта

BUF\_SIZE, // размер буфера

szName); // имя отраженного в памяти объекта

if (hMapFile == NULL || hMapFile == INVALID\_HANDLE\_VALUE)

{

printf("Не может создать отраженный в памяти объект (%ld).\n",

GetLastError());

return 1;

}

pBuf = (LPTSTR)MapViewOfFile(hMapFile, //дескриптор проецируемого в памяти объекта

FILE\_MAP\_ALL\_ACCESS, // разрешение чтения/записи(режим доступа)

0, //Старшее слово смещения файла, где начинается отображение

0, //Младшее слово смещения файла, где начинается отображение

BUF\_SIZE); //Число отображаемых байтов файла

if (pBuf == NULL)

{

printf("Представление проецированного файла невозможно (%ld).\n",

GetLastError());

return 2;

}

int i = 0;

while (true)

{

i = rand();

itoa(i, (char \*)szMsg, 10);

WaitForSingleObject(mutex, INFINITE);

CopyMemory((PVOID)pBuf, szMsg, sizeof(szMsg));

printf("write message: %s\n", (char \*)pBuf);

Sleep(1000); //необходимо только для отладки - для удобства представления и анализа //результатов

ReleaseMutex(mutex);

}

// освобождение памяти и закрытие описателя handle

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

CloseHandle(mutex);

}

Для организации синхронизации доступа к памяти в данном примере рассматривается использование мьютексов. В первом процессе создается именованный мьютекс, который "защищает" критические участки кода, в данном случае - запись в общую разделяемую память. Записываемая в буфер информация - это случайное число, которое генерируется функцией rand()перед каждой записью. Второй процесс открывает ранее созданный и именованный первым процессом мьютекс.

**Текст программы читателя:**

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#define BUF\_SIZE 256

#define TIME 15 // number of reading operation in this process

TCHAR szName[] = TEXT("MyFileMappingObject");

HANDLE WINAPI mutex;

int main()

{

HANDLE hMapFile;

LPCTSTR pBuf;

mutex = OpenMutex(

MUTEX\_ALL\_ACCESS, // request full access

FALSE, // handle not inheritable

TEXT("SyncMutex")); // object name

if (mutex == NULL)

printf("OpenMutex error: %ld\n", GetLastError());

else printf("OpenMutex successfully opened the mutex.\n");

hMapFile = OpenFileMapping(

FILE\_MAP\_ALL\_ACCESS, // доступ к чтению/записи

FALSE, // имя не наследуется

szName); // имя "проецируемого " объекта

if (hMapFile == NULL)

{

printf("Невозможно открыть объект проекция файла (%ld).\n", GetLastError());

return 1;

}

pBuf = (LPTSTR)MapViewOfFile(hMapFile, // дескриптор "проецируемого" объекта

FILE\_MAP\_ALL\_ACCESS, // разрешение чтения/записи

0,

0,

BUF\_SIZE);

if (pBuf == NULL)

{

printf("Представление проецированного файла (%ld) невозможно .\n", GetLastError());

return 2;

}

for (int i = 0; i<TIME; i++)

{

WaitForSingleObject(mutex, INFINITE);

printf("read message: %s\n", (char \*)pBuf);

ReleaseMutex(mutex);

}

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

return 0;

}

**Результат выполнения программы:**

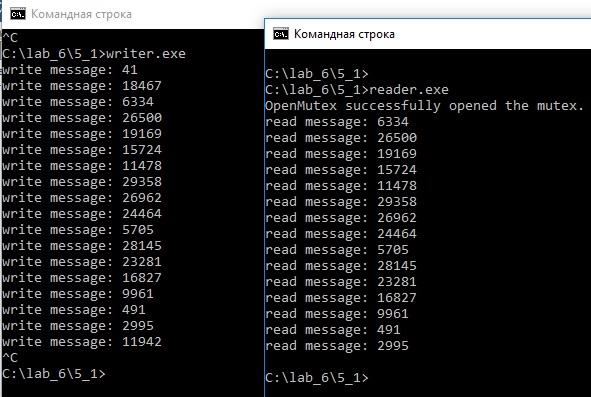


Рис.8. Пример работы программ с использованием разделяемой памяти

Первый процесс завершается по Ctrl+C, а второй считывает данные фиксированное количество раз (задаваемое переменной TIME).

**6. Почтовые слоты (MailSlot)**

MailSlot – механизм синхронизации, иначе называемый «почтовый ящик». Каждый слот реализуется как псевдофайл в оперативной памяти и содержит некоторое количество записей («сообщений»), которые могут быть прочтены всеми компьютерами в сетевом домене. Общий размер данных не может превышать 64K. В отличие от дисковых файлов, файлы MailSlot временные. Когда все указатели на MailSlot закрываются, MailSlot и все данные, которые он содержит, удаляются.

Для обмена посредством MailSlot создается клиент-серверное приложение. MailSlot сервер – является процессом, который создает и владеет MailSlot. При создании сервер получает указатель, используемый затем при чтении или записи данных им самим или другим процессом, получившим указатель на MailSlot. Создать слот можно только локально, а получать доступ (обращаться) и локально и удалённо. Для создания почтовых слотов используются имена в следующей форме: \\.\mailslot\[path]name . Форма имени при открытии почтового слота: \\ComputerName\mailslot\[path]name или \\DomainName\mailslot\[path]name

Для доступа по записи используют стандартные файловые операции получения дескриптора и работы с ним (CreateFile(), WriteFile(), CloseHandle()).

При создании почтовых слотов с одинаковым именем на нескольких компьютерах домена возможна широковещательная рассылка сообщений клиентов. Осуществление записи от множества клиентов может осуществляться одновременно. Сообщения располагаются в почтовом слоте в очереди в порядке записи до тех пор, пока поочередно (по одному за раз) не будут считаны (в той же очерёдности).

Для создания почтового слота используется функция:

HANDLE CreateMailslot {

LPCTSTR lpName, // имя

DWORD nMaxMessageSize, // максимальный размер

DWORD lReadTimeout, // интервал тайм-аута чтения LPSECURITY\_ATTRIBUTES lpSecurityAttributes // информация о безопасности

При успешном выполнении возвращается указатель, а в случае неудачи - INVALID\_HANDLE\_VALUE.

Для получения информации о наличии данных MailSlot, используеся функция:

BOOL GetMailslotInfo ( HANDLE hMailslot, // указатель на слот

LPDWORD lpMaxMessageSize,// максимальный размер

LPDWORD lpNextSize, // размер следующего

LPDWORD lpMessageCount, // количество сообщений

LPDWORD lpReadTimeout // тайм-аут. ); Возвращаемое значение равно 0, если слот не существует или произошла ошибка.

**6.1. Предложить собственную реализацию приложения, иллюстрирующую обмен информацией почтовыми слотами.**

**Исходный код сервера:**

#include "windows.h"

#include <stdio.h>

#include <conio.h>

#define g\_szMailslot "\\\\.\\mailslot\\ myslot" // имя почтового слота

#define BUFFER\_SIZE 1024

int main(int argc, char\* argv[])

{

HANDLE hMailslot;

//создаю почтовый слот

hMailslot = CreateMailslot(

g\_szMailslot, //имя слота

BUFFER\_SIZE, //размер входного буфера

MAILSLOT\_WAIT\_FOREVER, //отсутствие таймаута

NULL);

//обработка ошибки создания почтового слота

if (INVALID\_HANDLE\_VALUE == hMailslot) {

printf("\nError occurred while creating the mailslot: %d", GetLastError());

\_getch();

return 1; //Error

}

else

printf("\nCreateMailslot() was successful.");

//почтовые слоты - однонапревленное средство связи, так что сервер будет только считывать

char szBuffer[BUFFER\_SIZE];

DWORD cbBytes;

BOOL bResult;

printf("\nWaiting for client connection...");

while (1){

//читаем клиентское сообщение

bResult = ReadFile(

hMailslot,

szBuffer,

sizeof(szBuffer),

&cbBytes,

NULL);

//обработка возникновения ошибки при чтении

if ((!bResult) || (0 == cbBytes)){

printf("\nError occurred while reading "" from the client: %d", GetLastError());

CloseHandle(hMailslot);

return 1; //Error

}else{

printf("\nReadFile() was successful.");

}

printf("\nClient sent the following message: %s", szBuffer);

}

CloseHandle(hMailslot);

return 0;

}

**Исходный код клиента:**

#include <stdio.h>

#include "windows.h"

//Даю имя почтовому слоту

#define g\_szMailslot "\\\\.\\mailslot\\myslot"

#define BUFFER\_SIZE 1024

int main(int argc, char\* argv[])

{

HANDLE hMailslot;

//подсоединение к создавшему слот серверу

hMailslot = CreateFile(

g\_szMailslot, //имя слота

GENERIC\_WRITE, //слот только на запись (у сервера же - на чтение)

FILE\_SHARE\_READ,

NULL, //настройки безопасности дефолтные

OPEN\_EXISTING, //открывать существующий почтовый слот

FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (INVALID\_HANDLE\_VALUE == hMailslot){

printf("\nError occurred while connecting to the server: %d", GetLastError());

return 1; //Error

}else{

printf("\nCreateFile() was successful.");

}

//склиент только пишет

char szBuffer[BUFFER\_SIZE];

DWORD cbBytes;

while (true) {

printf("\nEnter a message to be sent to the server: ");

gets(szBuffer);

//посылка сообщения серверу

BOOL bResult = WriteFile(

hMailslot,

szBuffer, //буфер, откуда пишем

strlen(szBuffer) + 1, //сколько байт писать (+NULL)

&cbBytes, //сколько байт написали

NULL);

//обработчик ошибок

if ((!bResult) || (strlen(szBuffer) + 1 != cbBytes)){

printf("\nError occurred while writing"" to the server: %d", GetLastError());

CloseHandle(hMailslot);

return 1; //Error

}else{

printf("\nWriteFile() was successful.");

}

}

CloseHandle(hMailslot);

return 0; //Success

}

**Результат выполнения программы:**

Клиент использует функцию CreateFile с именем почтового слота в качестве имени файла.

Таким образом, клиент присоединяется к уже существующему почтовому слоту, созданному сервером. Запись данных в почтовый слот происходит при помощи функции WriteFile.

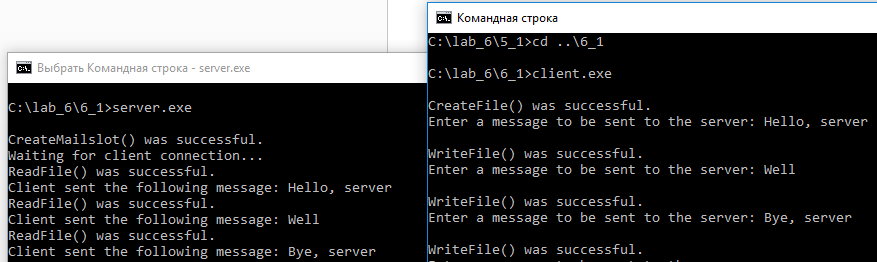


Рис.9. Пример работы программ с использованием MailSlot

**6.2. Продемонстрировать возможность локального и удаленного доступа.**

Для выполнения удаленного доступа, то есть использования почтовых слотов для передачи данных по сети, необходимо указать вместо точки в имени почтового слота адрес компьютера в сети.

C:\lab\_6\6\_1>hostname

DESKTOP-RR9VLT8

Cледовательно, в программе-клиенте нужно изменить \\.\mailslot\myslot на

\\DESKTOP-RR9VLT8\mailslot\myslot.

**4. Вывод**

В ходе выполнения данной лабораторной работы были рассмотрены различные средства межпроцессного взаимодействия в ОС Windows. Были изучены и продемонстрированы возможности неименованных и именнованных каналов, получены практические навыки по построению клиент-серверных приложений на основе сокетов, изучены сигналы в Windows и способы их обработки, а также рассмотрены взаимодействия с помощью разделяемой памяти и mailbox.

**Дополнения**

**1.1.**

Каналы в Windows монтируются по специальному пути «\\.\pipe\».

Содержимое этой директории до запуска программы:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.4806157912315212804

127 файлов 208 байт

0 папок 0 байт свободно

Содержимое этой директории после многократного запуска программы:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.5780.4052.8256022487080577265

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.4806157912315212804

01.01.1601 03:00 1 mojo.5780.4052.361581828086482274

01.01.1601 03:00 1 mojo.5780.1544.13505561841314848709

01.01.1601 03:00 1 mojo.5780.1544.16733167618832247802

01.01.1601 03:00 1 mojo.5780.1544.1003198358315141599

01.01.1601 03:00 1 mojo.5780.4052.2240839092882487937

01.01.1601 03:00 1 mojo.5780.1544.12824603774918728410

01.01.1601 03:00 1 mojo.5780.4052.14094993585225234564

01.01.1601 03:00 1 mojo.5780.1544.9032990981846343984

01.01.1601 03:00 1 mojo.5780.1544.4751917673085367424

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

135 файлов 216 байт

0 папок 0 байт свободно

**1.2.**

Аналогично пункту 1.1. посмотрим содержимое директории \\.\pipe\ до и после выполнения программы:

Содержимое до выполнения:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

135 файлов 216 байт

0 папок 0 байт свободно

Содержимое после выполнения:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

01.01.1601 03:00 1 mojo.5780.4052.11758961105794829856

136 файлов 217 байт

0 папок 0 байт свободно

**2.1.**

Именованные каналы не существуют постоянно и не могут, в отличие от Unix, быть созданы как специальные файлы в произвольной доступной для записи файловой системе, но имеют временные имена (освобождаемые после закрытия последней ссылки на них) и монтируются по специальному пути «\\.\pipe\» (то есть у канала под названием «foo» полное имя будет «\\.\pipe\foo»).

Сореджимое \\.\pipe\ до запуска программы:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

01.01.1601 03:00 1 mojo.5780.4052.11758961105794829856

136 файлов 217 байт

0 папок 0 байт свободно

После запуска программы:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

01.01.1601 03:00 1 mojo.5780.4052.11758961105794829856

**01.01.1601 03:00 1 $MyPipe$**

137 файлов 218 байт

0 папок 0 байт свободно

Канал с именем $MyPipe$ был успешно создан.

После отключения клиента от сервера канал удаляется:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

01.01.1601 03:00 1 mojo.5780.4052.11758961105794829856

136 файлов 217 байт

0 папок 0 байт свободно

**2.2.**

Запустим сервер и 3-ёх клиентов, после чего посмотрим на содержимое \\.\pipe\:

C:\Users>dir \\.\pipe\\

Том в устройстве \\.\pipe имеет метку NamedPipe

Содержимое папки \\.\pipe

01.01.1601 03:00 3 InitShutdown

01.01.1601 03:00 4 lsass

01.01.1601 03:00 3 ntsvcs

01.01.1601 03:00 3 scerpc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-39c-0

01.01.1601 03:00 3 epmapper

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-25c-0

01.01.1601 03:00 3 LSM\_API\_service

01.01.1601 03:00 3 eventlog

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-54-0

01.01.1601 03:00 3 spoolss

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-65c-0

01.01.1601 03:00 3 atsvc

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-4a0-0

01.01.1601 03:00 4 wkssvc

01.01.1601 03:00 3 trkwks

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-304-0

01.01.1601 03:00 1 vmware-usbarbpipe

01.01.1601 03:00 4 srvsvc

01.01.1601 03:00 1 vmware-authdpipe

01.01.1601 03:00 26 NvMessageBusBroadcast

01.01.1601 03:00 1 Winsock2\CatalogChangeListener-2ec-0

01.01.1601 03:00 3 {F685F498-BFDE-49D3-8132-D708F76B990E}

01.01.1601 03:00 1 PIPE\_EVENTROOT\CIMV2SCM EVENT PROVIDER

01.01.1601 03:00 3 CanonCAPT30

01.01.1601 03:00 7 MsFteWds

01.01.1601 03:00 1 SearchTextHarvester

01.01.1601 03:00 3 W32TIME\_ALT

01.01.1601 03:00 4 DAV RPC SERVICE

01.01.1601 03:00 1 TDLN-3828-41

01.01.1601 03:00 1 TDLN-9268-41

01.01.1601 03:00 1 9EA6F070-37D3-424D-84AB-39C2C9FB0621\_8

01.01.1601 03:00 1 mojo.10164.7648.17688878374998513921

…

…

…

01.01.1601 03:00 1 mojo.5780.1544.6431561190568615142

01.01.1601 03:00 1 mojo.5780.4052.11758961105794829856

**01.01.1601 03:00 4 $$MyPipe$$**

137 файлов 221 байт

0 папок 0 байт свободно

Созданный канал занял не 1 байт, как каналы в предыдущих случаях, а 4 байта.

**3.1.**

После запуска программы-сервера введём команду netstat -ano и найдём подключение с заданным портом(7500):

C:\>netstat -ano

Активные подключения

Имя Локальный адрес Внешний адрес Состояние PID

TCP 0.0.0.0:135 0.0.0.0:0 LISTENING 924

TCP 0.0.0.0:445 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:902 0.0.0.0:0 LISTENING 2284

TCP 0.0.0.0:912 0.0.0.0:0 LISTENING 2284

TCP 0.0.0.0:1309 0.0.0.0:0 LISTENING 2264

TCP 0.0.0.0:45769 0.0.0.0:0 LISTENING 5612

TCP 0.0.0.0:49664 0.0.0.0:0 LISTENING 604

TCP 0.0.0.0:49665 0.0.0.0:0 LISTENING 84

TCP 0.0.0.0:49666 0.0.0.0:0 LISTENING 1628

TCP 0.0.0.0:49667 0.0.0.0:0 LISTENING 1184

TCP 0.0.0.0:49668 0.0.0.0:0 LISTENING 772

TCP 0.0.0.0:49669 0.0.0.0:0 LISTENING 748

TCP 127.0.0.1:6463 0.0.0.0:0 LISTENING 7664

**TCP 127.0.0.1:7500 0.0.0.0:0 LISTENING 9144**

TCP 127.0.0.1:51700 0.0.0.0:0 LISTENING 1248

…

…

После запуска программы-клиента происходят следующие изменения:

C:\>netstat -ano

Активные подключения

Имя Локальный адрес Внешний адрес Состояние PID

TCP 0.0.0.0:135 0.0.0.0:0 LISTENING 924

TCP 0.0.0.0:445 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:902 0.0.0.0:0 LISTENING 2284

TCP 0.0.0.0:912 0.0.0.0:0 LISTENING 2284

TCP 0.0.0.0:1309 0.0.0.0:0 LISTENING 2264

TCP 0.0.0.0:45769 0.0.0.0:0 LISTENING 5612

TCP 0.0.0.0:49664 0.0.0.0:0 LISTENING 604

TCP 0.0.0.0:49665 0.0.0.0:0 LISTENING 84

TCP 0.0.0.0:49666 0.0.0.0:0 LISTENING 1628

TCP 0.0.0.0:49667 0.0.0.0:0 LISTENING 1184

TCP 0.0.0.0:49668 0.0.0.0:0 LISTENING 772

TCP 0.0.0.0:49669 0.0.0.0:0 LISTENING 748

TCP 127.0.0.1:6463 0.0.0.0:0 LISTENING 7664

**TCP 127.0.0.1:7500 0.0.0.0:0 LISTENING 9144**

**TCP 127.0.0.1:7500 127.0.0.1:52309 ESTABLISHED 9144**

**…**

**…**

Видно, что сервер установил соединение.

Клиент также установил соединеие:

C:\>netstat -ano

Активные подключения

Имя Локальный адрес Внешний адрес Состояние PID

TCP 0.0.0.0:135 0.0.0.0:0 LISTENING 924

…

…

TCP 127.0.0.1:7500 0.0.0.0:0 LISTENING 9144

TCP 127.0.0.1:7500 127.0.0.1:52309 ESTABLISHED 9144

TCP 127.0.0.1:51700 0.0.0.0:0 LISTENING 1248

**TCP 127.0.0.1:52309 127.0.0.1:7500 ESTABLISHED 6700**

По завершению программ соединение прерывается.

**3.2.**

При запуске 3-ёх клиентов:

C:\>netstat -ano

Активные подключения

Имя Локальный адрес Внешний адрес Состояние PID

TCP 0.0.0.0:135 0.0.0.0:0 LISTENING 924

TCP 0.0.0.0:445 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:902 0.0.0.0:0 LISTENING 2284

TCP 0.0.0.0:912 0.0.0.0:0 LISTENING 2284

TCP 0.0.0.0:1309 0.0.0.0:0 LISTENING 2264

**TCP 0.0.0.0:7500 0.0.0.0:0 LISTENING 1800**

TCP 0.0.0.0:45769 0.0.0.0:0 LISTENING 5612

TCP 0.0.0.0:49664 0.0.0.0:0 LISTENING 604

TCP 0.0.0.0:49665 0.0.0.0:0 LISTENING 84

TCP 0.0.0.0:49666 0.0.0.0:0 LISTENING 1628

TCP 0.0.0.0:49667 0.0.0.0:0 LISTENING 1184

TCP 0.0.0.0:49668 0.0.0.0:0 LISTENING 772

TCP 0.0.0.0:49669 0.0.0.0:0 LISTENING 748

TCP 127.0.0.1:6463 0.0.0.0:0 LISTENING 7664

**TCP 127.0.0.1:7500 127.0.0.1:52310 ESTABLISHED 1800**

**TCP 127.0.0.1:7500 127.0.0.1:52311 ESTABLISHED 1800**

**TCP 127.0.0.1:7500 127.0.0.1:52312 ESTABLISHED 1800**

Видно, что программа-сервер установила соединение с тремя клиентами, а также продолжает слушать.

Клиенты также установили соединение:

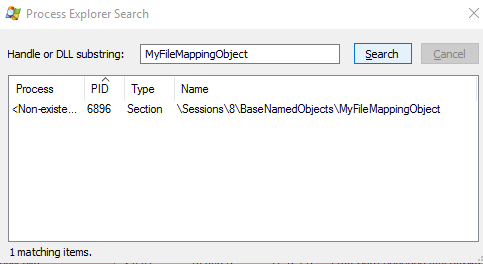
TCP 127.0.0.1:52310 127.0.0.1:7500 ESTABLISHED 4756

TCP 127.0.0.1:52311 127.0.0.1:7500 ESTABLISHED 8288

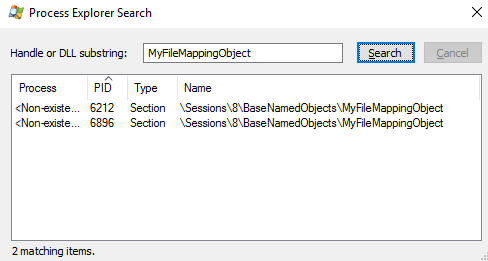
TCP 127.0.0.1:52312 127.0.0.1:7500 ESTABLISHED 1352

**5.1.**

При запуске программы-писателя создаётся Memory-mapped file с именем MyFileMappingObject:



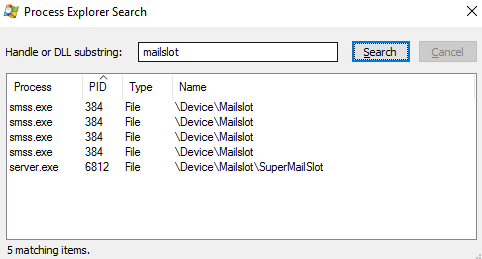
После запуска программы-читателя:



Видно, что созданный Memory-mapped file используют оба процесса.

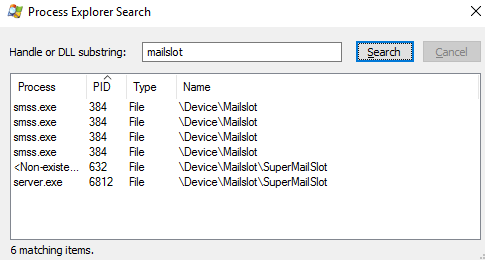
**6.1.**

Запустим программу сервер, откроем Process Explorer, затем, с помощью Find -> Find Handle or DLL найдём созданные почтовые слоты:



Видим, что создан mailslot с заданным именем.

После подключения клиента:



После завершения работы программы:

