МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

**ЛАБОРАТОРНАЯ РАБОТА** **№11**

**ИМЕННОВАННЫЕ КАНАЛЫ (PIPES)**

Выполнили: Карпович П.И., Ральцевич Э.В., Коляго Л.В. ФИТ, 3 курс, 5 группа(ПОИТ)

Проверила: Герман Ю.О.

2017

Сервер:

#include "stdafx.h"

#include <Windows.h>

#include <string.h>

int \_tmain(int argc, \_TCHAR\* argv[])

{

HANDLE hPipe;

char buffer[1024];

DWORD dwRead;

hPipe = CreateNamedPipe(TEXT("\\\\.\\pipe\\Pipe"),

PIPE\_ACCESS\_DUPLEX | PIPE\_TYPE\_BYTE | PIPE\_READMODE\_BYTE,

PIPE\_WAIT,

1,

1024 \* 16,

1024 \* 16,

NMPWAIT\_USE\_DEFAULT\_WAIT,

NULL);

while (hPipe != INVALID\_HANDLE\_VALUE)

{

if (ConnectNamedPipe(hPipe, NULL) != FALSE)

{

while (ReadFile(hPipe, buffer, sizeof(buffer) - 1, &dwRead, NULL) != FALSE)

{

buffer[dwRead] = '\0';

printf("%s", buffer);

}

}

DisconnectNamedPipe(hPipe);

}

return 0;

}

Клиент:

#include "stdafx.h"

#include <Windows.h>

int \_tmain(int argc, \_TCHAR\* argv[])

{

HANDLE hPipe;

DWORD dwWritten;

hPipe = CreateFile(TEXT("\\\\.\\pipe\\Pipe"),

GENERIC\_READ | GENERIC\_WRITE,

0,

NULL,

OPEN\_EXISTING,

0,

NULL);

if (hPipe != INVALID\_HANDLE\_VALUE)

{

WriteFile(hPipe,

"Hello Pipe\n",

12,

&dwWritten,

NULL);

CloseHandle(hPipe);

}

return (0);

}



**ЗАДАНИЕ**

Попробуйте выполнить передачу сообщения в ответ на принятое (реализовать двунаправленную передачу).

Сервер:

#include "stdafx.h"

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

#include <strsafe.h>

#define BUFSIZE 512

DWORD WINAPI InstanceThread(LPVOID);

VOID GetAnswerToRequest(LPTSTR, LPTSTR, LPDWORD);

int \_tmain(VOID)

{

BOOL fConnected = FALSE;

DWORD dwThreadId = 0;

HANDLE hPipe = INVALID\_HANDLE\_VALUE, hThread = NULL;

LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

for (;;)

{

\_tprintf(TEXT("\nPipe Server: Main thread awaiting client connection on %s\n"), lpszPipename);

hPipe = CreateNamedPipe(

lpszPipename, // pipe name

PIPE\_ACCESS\_DUPLEX, // read/write access

PIPE\_TYPE\_MESSAGE | // message type pipe

PIPE\_READMODE\_MESSAGE | // message-read mode

PIPE\_WAIT, // blocking mode

PIPE\_UNLIMITED\_INSTANCES, // max. instances

BUFSIZE, // output buffer size

BUFSIZE, // input buffer size

0, // client time-out

NULL); // default security attribute

if (hPipe == INVALID\_HANDLE\_VALUE)

{

\_tprintf(TEXT("CreateNamedPipe failed, GLE=%d.\n"), GetLastError());

return -1;

}

fConnected = ConnectNamedPipe(hPipe, NULL) ? TRUE : (GetLastError() == ERROR\_PIPE\_CONNECTED);

if (fConnected)

{

printf("Client connected, creating a processing thread.\n");

hThread = CreateThread(

NULL, // no security attribute

0, // default stack size

InstanceThread, // thread proc

(LPVOID)hPipe, // thread parameter

0, // not suspended

&dwThreadId); // returns thread ID

if (hThread == NULL)

{

\_tprintf(TEXT("CreateThread failed, GLE=%d.\n"), GetLastError());

return -1;

}

else CloseHandle(hThread);

}

else

CloseHandle(hPipe);

}

return 0;

}

DWORD WINAPI InstanceThread(LPVOID lpvParam)

{

HANDLE hHeap = GetProcessHeap();

TCHAR\* pchRequest = (TCHAR\*)HeapAlloc(hHeap, 0, BUFSIZE \* sizeof(TCHAR));

TCHAR\* pchReply = (TCHAR\*)HeapAlloc(hHeap, 0, BUFSIZE \* sizeof(TCHAR));

DWORD cbBytesRead = 0, cbReplyBytes = 0, cbWritten = 0;

BOOL fSuccess = FALSE;

HANDLE hPipe = NULL;

if (lpvParam == NULL)

{

printf("\nERROR - Pipe Server Failure:\n");

printf(" InstanceThread got an unexpected NULL value in lpvParam.\n");

printf(" InstanceThread exitting.\n");

if (pchReply != NULL) HeapFree(hHeap, 0, pchReply);

if (pchRequest != NULL) HeapFree(hHeap, 0, pchRequest);

return (DWORD)-1;

}

if (pchRequest == NULL)

{

printf("\nERROR - Pipe Server Failure:\n");

printf(" InstanceThread got an unexpected NULL heap allocation.\n");

printf(" InstanceThread exitting.\n");

if (pchReply != NULL) HeapFree(hHeap, 0, pchReply);

return (DWORD)-1;

}

if (pchReply == NULL)

{

printf("\nERROR - Pipe Server Failure:\n");

printf(" InstanceThread got an unexpected NULL heap allocation.\n");

printf(" InstanceThread exitting.\n");

if (pchRequest != NULL) HeapFree(hHeap, 0, pchRequest);

return (DWORD)-1;

}

printf("InstanceThread created, receiving and processing messages.\n");

hPipe = (HANDLE)lpvParam;

while (1)

{

fSuccess = ReadFile(

hPipe, // handle to pipe

pchRequest, // buffer to receive data

BUFSIZE \* sizeof(TCHAR), // size of buffer

&cbBytesRead, // number of bytes read

NULL); // not overlapped I/O

if (!fSuccess || cbBytesRead == 0)

{

if (GetLastError() == ERROR\_BROKEN\_PIPE)

{

\_tprintf(TEXT("InstanceThread: client disconnected.\n"), GetLastError());

}

else

{

\_tprintf(TEXT("InstanceThread ReadFile failed, GLE=%d.\n"), GetLastError());

}

break;

}

GetAnswerToRequest(pchRequest, pchReply, &cbReplyBytes);

fSuccess = WriteFile(

hPipe, // handle to pipe

pchReply, // buffer to write from

cbReplyBytes, // number of bytes to write

&cbWritten, // number of bytes written

NULL); // not overlapped I/O

if (!fSuccess || cbReplyBytes != cbWritten)

{

\_tprintf(TEXT("InstanceThread WriteFile failed, GLE=%d.\n"), GetLastError());

break;

}

}

FlushFileBuffers(hPipe);

DisconnectNamedPipe(hPipe);

CloseHandle(hPipe);

HeapFree(hHeap, 0, pchRequest);

HeapFree(hHeap, 0, pchReply);

printf("InstanceThread exitting.\n");

return 1;

}

VOID GetAnswerToRequest(LPTSTR pchRequest, LPTSTR pchReply, LPDWORD pchBytes)

{

\_tprintf(TEXT("Client Request String:\"%s\"\n"), pchRequest);

if (FAILED(StringCchCopy(pchReply, BUFSIZE, TEXT("default answer from server"))))

{

\*pchBytes = 0;

pchReply[0] = 0;

printf("StringCchCopy failed, no outgoing message.\n");

return;

}

\*pchBytes = (lstrlen(pchReply) + 1) \* sizeof(TCHAR);

}

Клиент:

#include "stdafx.h"

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#include <tchar.h>

#define BUFSIZE 512

int \_tmain(int argc, TCHAR \*argv[])

{

HANDLE hPipe;

LPTSTR lpvMessage = TEXT("Default message from client.");

TCHAR chBuf[BUFSIZE];

BOOL fSuccess = FALSE;

DWORD cbRead, cbToWrite, cbWritten, dwMode;

LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

if (argc > 1)

lpvMessage = argv[1];

while (1)

{

hPipe = CreateFile(

lpszPipename, // 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

if (hPipe != INVALID\_HANDLE\_VALUE)

break;

if (GetLastError() != ERROR\_PIPE\_BUSY) {

\_tprintf(TEXT("Could not open pipe. GLE=%d\n"), GetLastError());

return -1;

}

if (!WaitNamedPipe(lpszPipename, 20000))

{

printf("Could not open pipe: 20 second wait timed out.");

return -1;

}

}

dwMode = PIPE\_READMODE\_MESSAGE;

fSuccess = SetNamedPipeHandleState(

hPipe, // pipe handle

&dwMode, // new pipe mode

NULL, // don't set maximum bytes

NULL); // don't set maximum time

if (!fSuccess)

{

\_tprintf(TEXT("SetNamedPipeHandleState failed. GLE=%d\n"), GetLastError());

return -1;

}

cbToWrite = (lstrlen(lpvMessage) + 1) \* sizeof(TCHAR);

\_tprintf(TEXT("Sending %d byte message: \"%s\"\n"), cbToWrite, lpvMessage);

fSuccess = WriteFile(

hPipe, // pipe handle

lpvMessage, // message

cbToWrite, // message length

&cbWritten, // bytes written

NULL); // not overlapped

if (!fSuccess)

{

\_tprintf(TEXT("WriteFile to pipe failed. GLE=%d\n"), GetLastError());

return -1;

}

printf("\nMessage sent to server, receiving reply as follows:\n");

do

{

fSuccess = ReadFile(

hPipe, // pipe handle

chBuf, // buffer to receive reply

BUFSIZE \* sizeof(TCHAR), // size of buffer

&cbRead, // number of bytes read

NULL); // not overlapped

if (!fSuccess && GetLastError() != ERROR\_MORE\_DATA)

break;

\_tprintf(TEXT("\"%s\"\n"), chBuf);

} while (!fSuccess); // repeat loop if ERROR\_MORE\_DATA

if (!fSuccess)

{

\_tprintf(TEXT("ReadFile from pipe failed. GLE=%d\n"), GetLastError());

return -1;

}

CloseHandle(hPipe);

system("pause");

return 0;

}

