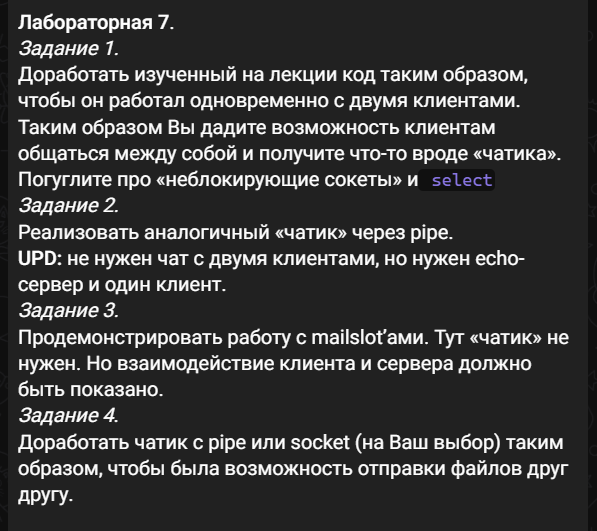
**Лабораторная работа 7**

**Студент:** Смирнов Константин ПКС-303

**Учитель:** Хусточка Алёна Витальевна



**Сервер**

include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <ws2def.h>

#include <Windows.h>

#include <ws2tcpip.h>

#include <wspiapi.h>

#include <processthreadsapi.h>

#pragma comment (lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

#define BUFFER\_SIZE 100

#define SIGNAL\_BYE "bye"

int result = E\_FAIL;

int count = 0;

WSADATA wsa = { 0 };

SOCKET listen\_socket = INVALID\_SOCKET;

unsigned long NonBlock = 1;

ADDRINFOA server = { 0 };

ADDRINFOA\* full\_server = NULL;

BOOL is\_running = TRUE;

SOCKET clients[2] = { INVALID\_SOCKET };

char buffer[BUFFER\_SIZE] = { 0 };

const char\* startbuffer = "Start chatting";

char startbuf[BUFFER\_SIZE] = { 0 };

int checker = 0;

DWORD bytesWritten;

bool flag = false;

DWORD WINAPI ServerThread(LPVOID param)

{

SOCKET\* client = (SOCKET\*)param;

int i = count;

int j = 0;

int newc = 0;

do {

result = recv(\*client, buffer, BUFFER\_SIZE, 0);

if (result > 0) {

if (!flag)

{

printf("Count of bytes received: %d\n", result);

printf("Client %d said: %s \n", i, buffer);

}

if (i == 2)

{

j = 0;

}

else

j = 1;

if (send(clients[j], buffer, result, 0) == SOCKET\_ERROR) {

result = 0;

}

if (strncmp(buffer, SIGNAL\_BYE, strlen(SIGNAL\_BYE)) == 0) {

send(clients[j], SIGNAL\_BYE, strlen(SIGNAL\_BYE), 0);

result = 0;

}

if (strncmp(buffer, "file", strlen("file")) == 0)

{

send(clients[j], "file", sizeof("file"), 0);

char file\_buffer[1024] = { 0 };

int result\_size = 0;

recv(\*client, (char\*)&result\_size, sizeof(result\_size), 0);

printf("size = %d\n", result\_size);

int current = 0;

int currentfile = 0;

send(clients[j], (char\*)&result\_size, sizeof(result\_size), 0);

do {

current = recv(\*client, file\_buffer, sizeof(file\_buffer), 0);

send(clients[j], file\_buffer, current, 0);

currentfile = current;

memset(file\_buffer, 0, sizeof(file\_buffer));

} while (current <= result\_size);

}

}

else if (result == 0) {

printf("Connection closing...\n");

}

else {

printf("Failed to recv with code %d\n", WSAGetLastError());

}

memset(buffer, 0, sizeof(buffer));

} while (result > 0);

result = shutdown(\*client, SD\_BOTH);

if (result == SOCKET\_ERROR) {

printf("Failed to shutdown with code %d\n", WSAGetLastError());

is\_running = FALSE;

}

closesocket(\*client);

printf("Client disconnected!\n\n");

return 0;

}

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

{

HANDLE ServerThreads[2] = { 0 };

const char\* PORT = "10203";

printf(" \n");

result = WSAStartup(0x0202, &wsa);

printf("WSAStartup returned: %d\n", result);

if (result != ERROR\_SUCCESS) {

goto cleanup;

}

server.ai\_family = AF\_INET;

server.ai\_socktype = SOCK\_STREAM;

server.ai\_protocol = IPPROTO\_TCP;

result = getaddrinfo("localhost", PORT, &server, &full\_server);

printf("Functoin getaddrinfo returned: %d\n", result);

if (result != ERROR\_SUCCESS) {

goto cleanup;

}

listen\_socket = socket(full\_server->ai\_family, full\_server->ai\_socktype, full\_server->ai\_protocol);

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

printf("Failed to create socket\n");

result = WSAGetLastError();

goto cleanup;

}

result = bind(listen\_socket, full\_server->ai\_addr, full\_server->ai\_addrlen);

if (result == SOCKET\_ERROR) {

goto cleanup;

}

result = listen(listen\_socket, SOMAXCONN);

if (result == SOCKET\_ERROR) {

goto cleanup;

}

printf("Create socket successfully\n\n");

while (is\_running) {

SOCKET client = INVALID\_SOCKET;

client = accept(listen\_socket, NULL, NULL);

if (client == INVALID\_SOCKET) {

printf("Failed to accept with code %d\n", WSAGetLastError());

continue;

}

if (count > 2)

{

printf("Too many people. Stop\n");

continue;

}

printf("New client connected!\n");

clients[count] = client;

ServerThreads[count] = CreateThread(NULL, 0, ServerThread, &clients[count], 0, 0);

count++;

if (count == 2)

{

if (checker == 0)

{

send(clients[0], startbuffer, 15, 0);

send(clients[1], startbuffer, 15, 0);

result = 0;

printf("Chatting should start\n");

checker = 1;

}

}

}

cleanup:

if (result != ERROR\_SUCCESS) {

printf("WSAGetLastError returned: %d\n", WSAGetLastError());

}

if (full\_server) freeaddrinfo(full\_server);

if (listen\_socket != INVALID\_SOCKET) closesocket(listen\_socket);

return result;

}

Client

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <Windows.h>

#include <ws2tcpip.h>

#include <wspiapi.h>

#pragma comment (lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

#define BUFFER\_SIZE 100

#define FILE\_BUFFER\_SIZE 1024

#define SIGNAL\_BYE "bye"

const char\* SIGNAL\_START = "Start chatting";

char buffer[MAX\_PATH] = { 0 };

SOCKET connect\_socket = INVALID\_SOCKET;

int result = E\_FAIL;

BOOL is\_running = FALSE;

HANDLE ClientSendThreads[1] = { 0 };

HANDLE ClientRecieveThreads[1] = { 0 };

DWORD bytesWritten;

DWORD bytesRead;

char readBuffer[BUFFER\_SIZE] = { 0 };

char size[BUFFER\_SIZE] = { 0 };

char readfileBuffer[FILE\_BUFFER\_SIZE] = { 0 };

BOOL isRecvFile = FALSE;

char q[10];

char savepath[1000] = { 0 };

bool flag = false;

wchar\_t\* CharToWchar(char\* str)

{

wchar\_t\* wstr = NULL;

int size = MultiByteToWideChar(CP\_ACP, 0, str, -1, NULL, 0);

wstr = (wchar\_t\*)malloc(size \* sizeof(wchar\_t));

MultiByteToWideChar(CP\_ACP, 0, str, -1, wstr, size \* sizeof(wchar\_t));

return wstr;

}

DWORD WINAPI ClientThreadSend(LPVOID param)

{

while (is\_running)

{

if (!isRecvFile)

{

printf("Enter message for server:\n");

fgets(buffer, sizeof(buffer), stdin);

result = send(connect\_socket, buffer, strlen(buffer), 0);

}

if (strncmp(buffer, SIGNAL\_BYE, strlen(SIGNAL\_BYE)) == 0)

{

printf("Server closed connection ):\n\n");

Sleep(800);

is\_running = FALSE;

result = ERROR\_SUCCESS;

CloseHandle(ClientSendThreads[1]);

CloseHandle(ClientRecieveThreads[1]);

}

if (strncmp(buffer, "file", strlen("file")) == 0)

{

char filename[1000] = { 0 };

printf("Enter file path: ");

scanf\_s("%s", filename, sizeof(filename));

wchar\_t\* filenameW = CharToWchar(filename);

//L"C:/Games/clonehero-win64/UnityPlayer.dll"

HANDLE file = CreateFile(filenameW, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, NULL, NULL);

free(filenameW);

int size = GetFileSize(file, NULL);

printf("size = %d\n", size);

send(connect\_socket, (char\*)&size, sizeof(size), 0);

int result\_size = 0;

do {

ReadFile(file, readfileBuffer, sizeof(readfileBuffer), &bytesRead, NULL);

send(connect\_socket, readfileBuffer, bytesRead, 0);

memset(readfileBuffer, 0, sizeof(readfileBuffer));

result\_size += bytesRead;

} while (result\_size <= size);

send(connect\_socket, readfileBuffer, size, 0);

}

if (result == SOCKET\_ERROR) {

printf("Failed to recieve data with code %d\n", WSAGetLastError());

is\_running = FALSE;

result = ERROR\_SUCCESS;

CloseHandle(ClientSendThreads[1]);

CloseHandle(ClientRecieveThreads[1]);

}

}

return 0;

}

DWORD WINAPI ClientThreadRecieve(LPVOID param)

{

while (is\_running) {

memset(buffer, 0, sizeof(buffer));

result = recv(connect\_socket, buffer, BUFFER\_SIZE, 0);

if (result > 0)

{

if (strncmp(buffer, SIGNAL\_BYE, strlen(SIGNAL\_BYE)) == 0) {

printf("Server closed connection ):\n\n");

Sleep(800);

is\_running = FALSE;

result = ERROR\_SUCCESS;

CloseHandle(ClientSendThreads[1]);

CloseHandle(ClientRecieveThreads[1]);

}

if (strncmp(buffer, "file", strlen("file")) == 0)

{

HANDLE file = CreateFile(L"test.txt", GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, CREATE\_ALWAYS, NULL, NULL);

int size\_result = 0;

int current = 0;

recv(connect\_socket, (char\*)&size\_result, sizeof(size\_result), 0);

do {

int temp = recv(connect\_socket, readfileBuffer, sizeof(readfileBuffer), 0);

WriteFile(file, readfileBuffer, temp, &bytesWritten, NULL);

SetFilePointer(file, NULL, NULL, FILE\_END);

memset(readfileBuffer, 0, sizeof(readfileBuffer));

current += temp;

} while (current < size\_result);

}

else if (result == SOCKET\_ERROR)

{

printf("Failed to recieve data with code %d\n", WSAGetLastError());

is\_running = FALSE;

result = ERROR\_SUCCESS;

CloseHandle(ClientSendThreads[1]);

CloseHandle(ClientRecieveThreads[1]);

}

else

{

printf("\nCount of bytes received: %d\n", result);

printf("Data received: %s\n", buffer);

}

}

}

return 0;

}

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

{

const char\* PORT = "10203";

printf(" \n");

WSADATA wsa = { 0 };

ADDRINFOA client = { 0 };

ADDRINFOA\* full\_client = NULL;

int i = 0;

int count = 1;

result = WSAStartup(0x0202, &wsa);

printf("WSAStartup returned: %d\n", result);

if (result != ERROR\_SUCCESS) {

goto cleanup;

}

client.ai\_family = AF\_INET;

client.ai\_socktype = SOCK\_STREAM;

client.ai\_protocol = IPPROTO\_TCP;

result = getaddrinfo("localhost", PORT, &client, &full\_client);

printf("Function getaddrinfo returned: %d\n", result);

if (result != ERROR\_SUCCESS) {

goto cleanup;

}

connect\_socket = socket(full\_client->ai\_family, full\_client->ai\_socktype, full\_client->ai\_protocol);

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

printf("Failed to create socket\n");

result = WSAGetLastError();

goto cleanup;

}

printf("Create socket successfully\n");

result = connect(connect\_socket, full\_client->ai\_addr, full\_client->ai\_addrlen);

if (result == SOCKET\_ERROR) {

goto cleanup;

}

else

{

memset(buffer, 0, sizeof(buffer));

result = recv(connect\_socket, buffer, BUFFER\_SIZE, 0);

printf("%s\n", buffer);

}

if (strncmp(buffer, SIGNAL\_START, strlen(SIGNAL\_START)) == 0)

{

memset(buffer, 0, sizeof(buffer));

is\_running = TRUE;

}

ClientSendThreads[1] = CreateThread(NULL, 0, ClientThreadSend, 0, 0, 0);

ClientRecieveThreads[1] = CreateThread(NULL, 0, ClientThreadRecieve, 0, 0, 0);

printf("thread created!\n");

WaitForSingleObject(ClientSendThreads[1], INFINITE);

cleanup:

if (result != ERROR\_SUCCESS) {

printf("WSAGetLastError returned: %d\n", WSAGetLastError());

}

if (full\_client) freeaddrinfo(full\_client);

if (connect\_socket != INVALID\_SOCKET) closesocket(connect\_socket);

return result;

}

**Pipe Server**

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

#include <strsafe.h>

#define CONNECTING\_STATE 0

#define READING\_STATE 1

#define WRITING\_STATE 2

#define INSTANCES 4

#define PIPE\_TIMEOUT 5000

#define BUFSIZE 4096

typedef struct

{

OVERLAPPED oOverlap;

HANDLE hPipeInst;

CHAR chRequest[BUFSIZE];

DWORD cbRead;

CHAR chReply[BUFSIZE];

DWORD cbToWrite;

DWORD dwState;

BOOL fPendingIO;

} PIPEINST, \* LPPIPEINST;

VOID DisconnectAndReconnect(DWORD);

BOOL ConnectToNewClient(HANDLE, LPOVERLAPPED);

VOID GetAnswerToRequest(LPPIPEINST);

PIPEINST Pipe[INSTANCES];

HANDLE hEvents[INSTANCES];

int \_tmain(VOID)

{

DWORD i, dwWait, cbRet, dwErr;

BOOL fSuccess;

LPCWSTR lpszPipename = L"\\\\.\\pipe\\Pipe";

printf("Server started\n");

for (i = 0; i < INSTANCES; i++)

{

hEvents[i] = CreateEvent(NULL, TRUE, TRUE, NULL);

if (hEvents[i] == NULL)

{

printf("CreateEvent failed with %d.\n", GetLastError());

return 0;

}

Pipe[i].oOverlap.hEvent = hEvents[i];

Pipe[i].oOverlap.Offset = 0;

Pipe[i].oOverlap.OffsetHigh = 0;

Pipe[i].hPipeInst = CreateNamedPipe(lpszPipename, PIPE\_ACCESS\_DUPLEX | FILE\_FLAG\_OVERLAPPED, PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

INSTANCES, BUFSIZE \* sizeof(TCHAR), BUFSIZE \* sizeof(TCHAR), PIPE\_TIMEOUT, NULL);

if (Pipe[i].hPipeInst == INVALID\_HANDLE\_VALUE)

{

printf("CreateNamedPipe failed with %d.\n", GetLastError());

return 0;

}

Pipe[i].fPendingIO = ConnectToNewClient(

Pipe[i].hPipeInst,

&Pipe[i].oOverlap);

Pipe[i].dwState = Pipe[i].fPendingIO ?

CONNECTING\_STATE :

READING\_STATE;

}

while (1)

{

dwWait = WaitForMultipleObjects(INSTANCES, hEvents, FALSE, INFINITE);

i = dwWait - WAIT\_OBJECT\_0;

if (i < 0 || i >(INSTANCES - 1))

{

printf("Index out of range.\n");

return 0;

}

if (Pipe[i].fPendingIO)

{

fSuccess = GetOverlappedResult(Pipe[i].hPipeInst, &Pipe[i].oOverlap, &cbRet, FALSE);

switch (Pipe[i].dwState)

{

case CONNECTING\_STATE:

if (!fSuccess)

{

printf("Error %d.\n", GetLastError());

return 0;

}

Pipe[i].dwState = READING\_STATE;

break;

case READING\_STATE:

if (!fSuccess || cbRet == 0)

{

DisconnectAndReconnect(i);

continue;

}

Pipe[i].cbRead = cbRet;

Pipe[i].dwState = WRITING\_STATE;

break;

case WRITING\_STATE:

if (!fSuccess || cbRet != Pipe[i].cbToWrite)

{

DisconnectAndReconnect(i);

continue;

}

Pipe[i].dwState = READING\_STATE;

break;

default:

{

printf("Invalid pipe state.\n");

return 0;

}

}

}

switch (Pipe[i].dwState)

{

case READING\_STATE:

fSuccess = ReadFile(

Pipe[i].hPipeInst,

Pipe[i].chRequest,

BUFSIZE \* sizeof(TCHAR),

&Pipe[i].cbRead,

&Pipe[i].oOverlap);

if (fSuccess && Pipe[i].cbRead != 0)

{

Pipe[i].fPendingIO = FALSE;

Pipe[i].dwState = WRITING\_STATE;

continue;

}

dwErr = GetLastError();

if (!fSuccess && (dwErr == ERROR\_IO\_PENDING))

{

Pipe[i].fPendingIO = TRUE;

continue;

}

DisconnectAndReconnect(i);

break;

case WRITING\_STATE:

GetAnswerToRequest(&Pipe[i]);

fSuccess = WriteFile(

Pipe[i].hPipeInst,

Pipe[i].chReply,

Pipe[i].cbToWrite,

&cbRet,

&Pipe[i].oOverlap);

if (fSuccess && cbRet == Pipe[i].cbToWrite)

{

Pipe[i].fPendingIO = FALSE;

Pipe[i].dwState = READING\_STATE;

continue;

}

dwErr = GetLastError();

if (!fSuccess && (dwErr == ERROR\_IO\_PENDING))

{

Pipe[i].fPendingIO = TRUE;

continue;

}

DisconnectAndReconnect(i);

break;

default:

{

printf("Invalid pipe state.\n");

return 0;

}

}

}

return 0;

}

VOID DisconnectAndReconnect(DWORD i)

{

if (!DisconnectNamedPipe(Pipe[i].hPipeInst))

{

printf("DisconnectNamedPipe failed with %d.\n", GetLastError());

}

Pipe[i].fPendingIO = ConnectToNewClient(

Pipe[i].hPipeInst,

&Pipe[i].oOverlap);

Pipe[i].dwState = Pipe[i].fPendingIO ?

CONNECTING\_STATE :

READING\_STATE;

}

BOOL ConnectToNewClient(HANDLE hPipe, LPOVERLAPPED lpo)

{

BOOL fConnected, fPendingIO = FALSE;

fConnected = ConnectNamedPipe(hPipe, lpo);

if (fConnected)

{

printf("ConnectNamedPipe failed with %d.\n", GetLastError());

return 0;

}

switch (GetLastError())

{

case ERROR\_IO\_PENDING:

fPendingIO = TRUE;

break;

case ERROR\_PIPE\_CONNECTED:

if (SetEvent(lpo->hEvent))

break;

default:

{

printf("ConnectNamedPipe failed with %d.\n", GetLastError());

return 0;

}

}

return fPendingIO;

}

VOID GetAnswerToRequest(LPPIPEINST pipe)

{

printf(("[%d] %s\n"), pipe->hPipeInst, pipe->chRequest);

StringCchCopyA(pipe->chReply, BUFSIZE, pipe->chRequest);

pipe->cbToWrite = (strlen(pipe->chReply) + 1) \* sizeof(TCHAR);

}

Pipe client

#include <stdio.h>

#include <Windows.h>

#include <conio.h>

#include <filesystem>

#include <tchar.h>

#include <processthreadsapi.h>

HANDLE hPipe;

DWORD dwWritten;

DWORD dwRead;

char buffer[1024];

HANDLE ClientSendThreads[1] = { 0 };

HANDLE ClientRecieveThreads[1] = { 0 };

DWORD WINAPI ClientThreadSend(LPVOID param)

{

while (1)

{

printf("Enter message for server:\n");

fgets(buffer, sizeof(buffer), stdin);

WriteFile(hPipe, buffer, sizeof(buffer) - 1, &dwWritten, NULL);

ReadFile(hPipe, buffer, sizeof(buffer) - 1, &dwRead, NULL);

buffer[dwRead] = '\0';

printf("Client: %s\n", buffer);

}

return 0;

}

int main(void)

{

hPipe = CreateFile(TEXT("\\\\.\\pipe\\Pipe"), GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

ClientSendThreads[0] = CreateThread(NULL, 0, ClientThreadSend, 0, 0, 0);

WaitForSingleObject(ClientSendThreads[0], INFINITE);

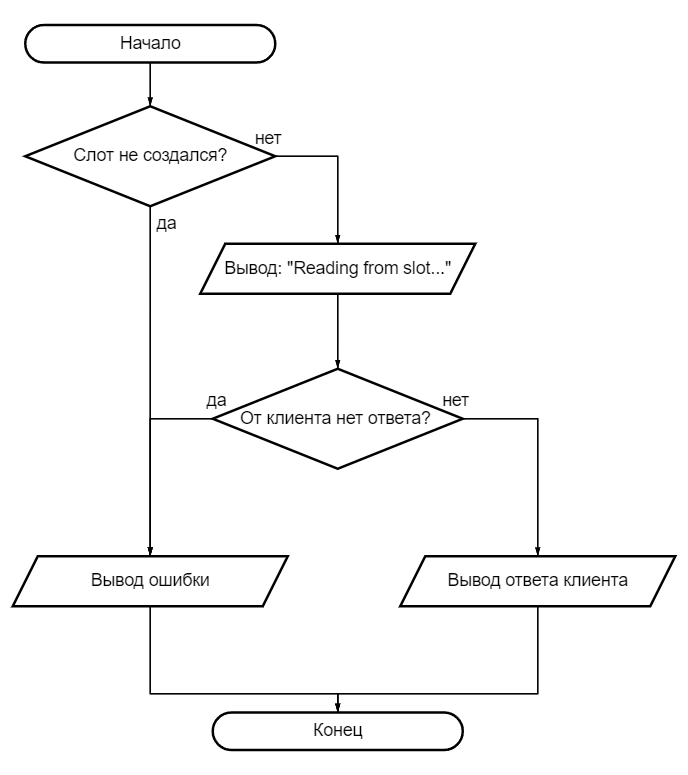
CloseHandle(hPipe);

\_getch();

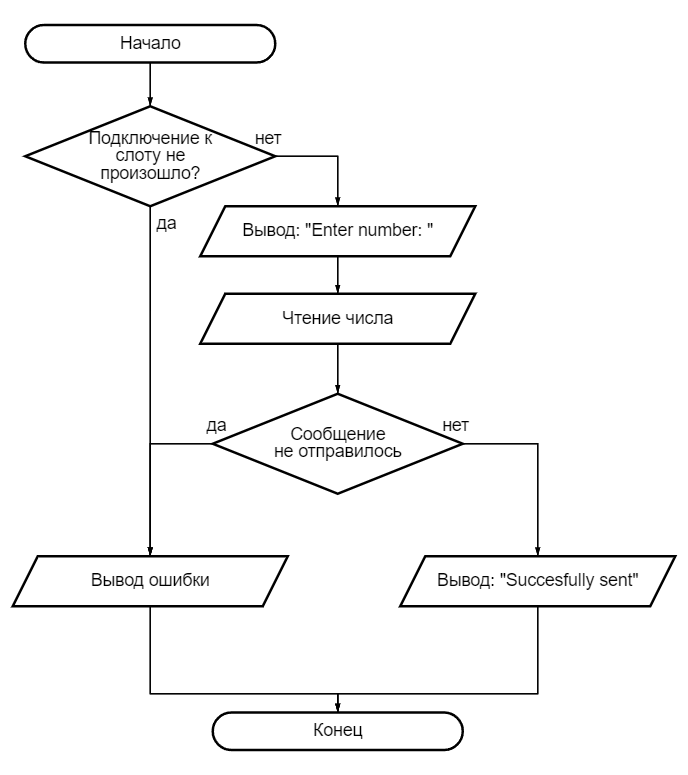
return (0);

}

**Блок-Схема Mailslot Сервера**



**Блок-Схема Mailslot клиента**

****

