Лабораторная работа №6

студента группы ИТ – 32

Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

МЕЖПРОЦЕССОРНОЕ ВЗАИМОДЕЙСТВИЕ. СЛУЖБЫ WINDOWS

Цель работы: изучение модели безопасности Windows. Получение практических навыков применения функций Win32 API для управления безопасностью Windows.

Содержание работы

Вариант 11

1. Разработать в Visual C++ оконное приложение Win32, которое, будучи запущенно несколько раз, способно выполнять обмен данными между всеми своими экземплярами.

Обмен данными должен выполняться с помощью механизма межпроцессного взаимодействия, указанного в варианте задания.

2. Разработать в Visual C++ два приложения:

- серверное приложение (реализованное, как служба Windows), которое хранит список студентов и обрабатывает запросы на получение данных о них;
- клиентское приложение (реализованное, как приложение Win32), которое запрашивает данные о студентах у серверного приложения и выводит полученные данные.

Обмен данными должен выполняться с помощью механизма межпроцессорного взаимодействия, указанного в варианте задания.

3. Протестировать работу разработанных приложений на компьютере под управлением Windows. Результаты отразить в отчете.

4. Включить в отчет исходный программный код и выводы о проделанной работе.

№	Механизм межпроцессного взаимодействия (п. 1 задания)	Механизм межпроцессного взаимодействия (п. 2 задания)
11,26	Оконные сообщения	Именованные каналы

Рис. 6.1. задание для варианта 11

Ход работы

1. В соответствии с п.1. разработала приложение Win32, осуществляющего указанные действия.

Листинг 1. Приложение

```
	#define MAX_MESSAGE_SIZE		512
	#define MAX_USERNAME_SIZE			20

	/*Дескрипторы*/
	HWND hwnd = NULL; // дескриптор главного окна
	HKEY hKey = NULL; // дескриптор открытого ключа реестра для запущенных экзм
	HANDLE hThreads[2]S; // дескрипторы созданных потоков

	/*Оконные процедуры*/
	LRESULT CALLBACK MainWindowProcess(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

	BOOL PreTranslateMessage(LPMSG lpMsg);//вызывает перед передачей сообщений в оконную процедуру

	/*Обработчики WM_CREATE, WM_DESTROY, WM_SIZE*/

	BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct);
	void OnDestroy(HWND hwnd);


	/*---------Работа с реестром---------------*/
	BOOL RegisterApplication();// регистрация каждого нового экземпляра приложения
	void UregisterApplication();// удаление информации о каждом созданном экземпляре
	/*-------------------------------------*/

	/*-----Отправка оконных сообщений-------*/
	void SendText(LPCTSTR lpText, DWORD cchText, BOOL fCopyData);// отправка с помощью оконных
сообщений
	void StartChat(HWND hwnd, LPCTSTR lpszText);// обработчик события WM_SETTEXT




	int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR CmdLine, int CmdShow)
	{

		/*Регистрация оконного класса и обработка ошибки*/
		WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

		wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
		wcex.lpfnWndProc = MainWindowProcess; // оконная процедура
		wcex.hInstance = hInstance;
		wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
		wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
		wcex.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(0, 100, 256));
		wcex.lpszClassName = TEXT("MainWindowProcess"); // имя класса
		wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

		if (0 == RegisterClassEx(&wcex))
		{
			return -1;
		}
		/*-------------------------------------------*/
		LoadLibrary(TEXT("ComCtl32.dll"));//

		/*Создание главного файла и обработка ошибки */
		hwnd = CreateWindowEx(0, TEXT("MainWindowProcess"), TEXT("Chat"),
			WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, 520, 520, NULL, NULL, hInstance, NULL);

		if (hwnd == NULL)
		{
			return -1;
		}
```

```
	/*-----------------------------------------------*/
	/*Регистрация в системном реестре и обработка ошибки*/
	BOOL RetRes = RegisterApplication();// регистрируем приложение в системном реестре
	if (RetRes == FALSE)
	{
		return -1;
	}
	/*-----------------------------------------------*/

	ShowWindow(hwnd, CmdShow); // отображаем главное окно

	/*Цикл обработки сообщений*/
	MSG  msg;

	while ((RetRes = GetMessage(&msg, NULL, 0, 0)) != FALSE)
	{
		if (RetRes == -1)
		{
			//Error editing
		}
		else if (!PreTranslateMessage(&msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	UregisterApplication();// удаляем информацию о приложении из системного реестра
	return (int)msg.wParam;
}


/*Процедура главного окна*/
LRESULT CALLBACK MainWindowProcess(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
	switch (msg)
	{
		HANDLE_MSG(hWnd, WM_CREATE, OnCreate);
		HANDLE_MSG(hWnd, WM_DESTROY, OnDestroy);
		HANDLE_MSG(hWnd, WM_SETTEXT, StartChat);
	}
	return DefWindowProc(hWnd, msg, wParam, lParam);
}

/*Обработчик сообщений*/
BOOL PreTranslateMessage(LPMSG Msg)
{
	/*Переменные*/
	TCHAR Message[MAX_MESSAGE_SIZE]; //сообщение
	TCHAR UserName[MAX_USERNAME_SIZE];//имя отправителя
	TCHAR UserMessage[532] = _T("");

	DWORD symbols, symb_user;  //количество символов в сообщении

	if ((WM_KEYDOWN == Msg->message) && (VK_RETURN == Msg->wParam)) // нажата клавиша Enter
	{
		HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_TEXT);
		HWND hwndUser = GetDlgItem(hwnd, IDC_EDIT_USERNAME);

		if (GetFocus() == hwndCtl) // поле ввода обладает фокусом клавиатуры
		{
			/*Чтобы можно было отправить многострочный текст*/
			/*CTRL+ENTER*/
			if (GetKeyState(VK_SHIFT) < 0) // нажата клавиша SHIFT
			{
				Edit_ReplaceSel(hwndCtl, _T("\r\n"));
			}
			else
			{
				symbols  =  Edit_GetText(hwndCtl,  Message,  _countof(Message));//
копируем текст из поля ввода
```

```
                                    symb_user = Edit_GetText(hwndUser, UserName, _countof(UserName));//
копируем текст из поля ввода

                                    if (symbols > 0)
                                    {
                                           // очищаем поле ввода
                                           Edit_SetText(hwndCtl, NULL);

                                           StringCchCat(UserName, _countof(UserName), _T(":"));
                                           StringCchCat(Message, _countof(Message), _T("\n\n"));

                                           StringCchCat(UserMessage, _countof(UserMessage), UserName);

                                           StringCchCat(UserMessage, _countof(UserMessage), Message);

                                           SendText(UserMessage, _tcslen(UserMessage), FALSE);

                                    }
                             }
                             return TRUE;
                      }
               }
               return FALSE;
        }

        BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct)
        {
               // создаём событие для уведомления потока ThreadFuncClipboard о завершения приложения
               hStopper = CreateEventEx(NULL, NULL, CREATE_EVENT_MANUAL_RESET, EVENT_ALL_ACCESS);


               DWORD dwStyle = WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_HSCROLL | ES_MULTILINE |
ES_AUTOHSCROLL | ES_AUTOVSCROLL;

               // создаём поле вывода сообщений
               CreateWindowEx(WS_EX_STATICEDGE, TEXT("Edit"), TEXT(""), dwStyle | ES_READONLY,
                      10, 10, 490, 250, hwnd, (HMENU)IDC_EDIT_MESSAGES, lpCreateStruct->hInstance, NULL);

               //Для username
               CreateWindowEx(0, TEXT("Static"), TEXT("Имя: "), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
                      10, 270, 40, 40, hwnd, NULL, lpCreateStruct->hInstance, NULL);

               CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("Edit"), TEXT(""), WS_CHILD | WS_VISIBLE ,
                      50, 270, 450, 40, hwnd, (HMENU)IDC_EDIT_USERNAME, lpCreateStruct->hInstance, NULL);

               // создаём поле ввода
               HWND hwndCtl = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("Edit"), TEXT(""), dwStyle,
                      10, 320, 490, 140, hwnd, (HMENU)IDC_EDIT_TEXT, lpCreateStruct->hInstance, NULL);

               // задаём ограничение на ввод текста
               Edit_LimitText(hwndCtl, MAX_MESSAGE_SIZE);
               // передаём фокус полю ввода
               //SetFocus(hwndCtl);

               return TRUE;
        }

        void OnDestroy(HWND hwnd)
        {
               /*После закрытия окна приложения*/

               SetEvent(hStopper);//  завершить поток ThreadFuncClipboard
               WaitForMultipleObjects(_countof(hThreads),  hThreads,  TRUE,  INFINITE);//ждем  заверения
остальных потоков
               CloseHandle(hStopper);// закрываем дескриптор события для завершении приложения
               PostQuitMessage(0); // отправляем сообщение WM_QUIT
        }


        void StartChat(HWND hwnd, LPCTSTR Message)
        {
```

```
		HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_MESSAGES);
		Edit_SetSel(hwndCtl, Edit_GetTextLength(hwndCtl), -1);// устанавливаем курсор в конец поля
вывода
		Edit_ReplaceSel(hwndCtl, Message);// вставляем текст в поле вывода
	}


	BOOL RegisterApplication()
	{
		/*Переменные*/
		DWORD lpdwDisposition;	//Адрес переменной, куда будет возвращено: Ключ не существует
и был создан/ Ключ был открыт
		DWORD PID;			//идентификатор процесса

		LONG lStatus;		//результат создания ключа
		HKEY  SubKey_Handle = NULL;		// дескриптор вложенного ключа реестра, который будет
содержать данные текущего приложения
		TCHAR SubKey_Name[20];			// имя вложенного ключа

		// создаём и открываем ключ реестра HKEY_CURRENT_USER\Software\\IT-311

		lStatus = RegCreateKeyEx(HKEY_CURRENT_USER, TEXT("Software\\IT-311"),
			0, NULL, REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, &lpdwDisposition);

		if (lStatus == ERROR_SUCCESS)
		{
			PID = GetCurrentProcessId();// получим идентификатор текущего процесса

			StringCchPrintf(SubKey_Name, _countof(SubKey_Name), TEXT("%d"), PID);// формируем
имя вложенного ключа реестра

			lStatus = RegCreateKey(hKey, SubKey_Name, &SubKey_Handle);// создаём вложенный ключ
реестра

			if (ERROR_SUCCESS == lStatus)
			{
				RegSetValueEx(SubKey_Handle,	NULL,	0,	REG_DWORD,	(LPBYTE)&PID,
sizeof(DWORD));// сохраняем идентификатор текущего процесса
				RegSetValueEx(SubKey_Handle,  TEXT("hwnd"),  0,  REG_BINARY,  (LPBYTE)&hwnd,
sizeof(HWND));// сохраняем значение дескриптор главного окна
				RegCloseKey(SubKey_Handle), SubKey_Handle = NULL; // закрываем вложенный
ключ реестра
				return TRUE;
			}
			RegCloseKey(hKey), hKey = NULL;// закрываем ключ реестра
		}
		return FALSE;
	}

	void UregisterApplication()
	{
		/*Переменные*/
		DWORD SubKeys_quant = 0; // количество вложенных ключей
		TCHAR SubKey_Name[20]; // имя вложенного ключа реестра, который содержит данные текущего
приложения
		LSTATUS lStatus; //результат определения количества ключей
		if (hKey != NULL)
		{
			lStatus = RegQueryInfoKey(hKey, NULL, NULL, NULL, &SubKeys_quant, NULL, NULL, NULL,
NULL, NULL, NULL, NULL);

			if ((lStatus == ERROR_SUCCESS) && (SubKeys_quant < 2))
			{
				RegDeleteTree(hKey, NULL);// удаляем всю ветку
				RegDeleteKey(HKEY_CURRENT_USER, TEXT("Software\\IT-311"));// удаляем ключ
			}
			else
			{
				StringCchPrintf(SubKey_Name,	_countof(SubKey_Name),	TEXT("%d"),
GetCurrentProcessId());	// формируем имя вложенного ключа реестра
				RegDeleteKey(hKey, SubKey_Name);	// удаляем вложенный ключ
```

```
                }
                RegCloseKey(hKey), hKey = NULL;     // закрываем ключ реестра
            }
        }

        void SendText(LPCTSTR Message, DWORD Message_Size, BOOL DataCopy)
        {
            /*-------Переменнные-------*/

            //дескрипторы
            HWND hRecvWnd; // дескриптор окна получателя
            HKEY hSubKey = NULL; // дескриптор вложенного ключа реестра

            //для  вложенного ключа
            TCHAR SubKey_Name[20]; // имя вложенного ключа реестра
            DWORD SubKey_Name_sz; // для определения размера имени ключа
            DWORD SubKeys_quant = 0; // количество вложенных ключей
            LSTATUS lStatus; //для результатов с вложенными ключами

            DWORD lpcbData;//адрес переменной для размера буфера данных

            if ((Message != NULL) && (Message_Size > 0))
            {
                lStatus = RegQueryInfoKey(hKey, NULL, NULL, NULL, &SubKeys_quant, NULL, NULL, NULL,
NULL, NULL, NULL, NULL);// определяем количество вложенных ключей

                if ((ERROR_SUCCESS == lStatus) && (SubKeys_quant > 0))
                {
                    //начало перечисления ключей
                    for (DWORD i = 0; i < SubKeys_quant; ++i)
                    {
                        SubKey_Name_sz = _countof(SubKey_Name);
                        lStatus = RegEnumKeyEx(hKey, i, SubKey_Name, &SubKey_Name_sz, NULL,
NULL, NULL, NULL);//получим имя ключа по индексу

                        if (lStatus == ERROR_SUCCESS)
                        {
                            lStatus = RegOpenKeyEx(hKey, SubKey_Name, 0, KEY_QUERY_VALUE,
&hSubKey);// открываем вложенный ключ
                        }

                        if (lStatus == ERROR_SUCCESS)
                        {

                            lpcbData = sizeof(HWND);

                            lStatus = RegQueryValueEx(hSubKey, TEXT("hwnd"), NULL, NULL,
(LPBYTE)&hRecvWnd, &lpcbData);// получаем дескриптор окна получателя

                            if (lStatus == ERROR_SUCCESS)
                            {
                                SendMessage(hRecvWnd,          WM_SETTEXT,          0,
(LPARAM)Message);// отправляем сообщение WM_SETTEXT
                            }
                            RegCloseKey(hSubKey), hSubKey = NULL;// закрываем вложенный
ключ реестра
                        }
                    }
                }
            }
        }
```

2. Разработала программу в соответствии с указанным в п.2 заданием.

Листинг 2. Код описывающий службу WinService

```
#include <Windows.h>
```

```
#include <stdio.h>
#include <tchar.h>
#include <locale.h>
#include <strsafe.h>
#include <fstream>
#include <iostream>

LPCTSTR service_name = TEXT("DemoService"); //// внутреннее имя сервиса, используемое SCM
LPCTSTR SvcDisplayName = TEXT("DemoService");// внешнее имя сервиса в панели управления

SERVICE_STATUS service_status; // текущее состояние службы
SERVICE_STATUS_HANDLE hServiceStatus; // дескриптор состояния службы

std::ofstream  out;   // выходной файл для протокола работы сервиса
int  nCount;     // счетчик

// ------------------------------------------------------------------------------------------------
--

BOOL OnSvcInit(DWORD dwArgc, LPTSTR *lpszArgv);// эта функция вызывается при запуске службы

void OnSvStop(void);// эта функция вызывается для остановки службы

DWORD SvcMain(DWORD dwArgc, LPTSTR *lpszArgv);// в этой функции реализован основной функционал


DWORD WINAPI SvcHandler(DWORD fdwControl, DWORD dwEventType, LPVOID lpEventData, LPVOID
lpContext)
{
	if (SERVICE_CONTROL_STOP == fdwControl || SERVICE_CONTROL_SHUTDOWN == fdwControl)
	{
		OnSvcStop(); // останавливаем службу
		service_status.dwCurrentState = SERVICE_STOP_PENDING; // новое состояние службы
	}

	SetServiceStatus(hServiceStatus, &service_status);// изменяем текущее состояние службы
	return NO_ERROR;
}

// ------------------------------------------------------------------------------------------------
--
void WINAPI ServiceMain(DWORD dwArgc, LPTSTR *lpszArgv)
{
	// регистрируем обработчик управляющих команд для сервиса
	hServiceStatus = RegisterServiceCtrlHandlerEx(service_name, SvcHandler, NULL);

	if (!hServiceStatus)
	{
		out.open("C:\\ServiceFile.log");
		out << "Register service control handler failed.";
		out.close();

		return;
	}

	if (NULL != hServiceStatus)
	{
		// начальное состояние
		service_status.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
		service_status.dwCurrentState = SERVICE_START_PENDING;
		service_status.dwControlsAccepted = SERVICE_ACCEPT_STOP | SERVICE_ACCEPT_SHUTDOWN;
		service_status.dwWin32ExitCode = NO_ERROR;
		service_status.dwServiceSpecificExitCode = 0;
		service_status.dwCheckPoint = 0;
		service_status.dwWaitHint = 0;

		// задаем начальное состояние службы
```

```
		SetServiceStatus(hServiceStatus, &service_status);


		if (OnSvcInit(dwArgc, lpszArgv) != FALSE)
		{
			service_status.dwCurrentState = SERVICE_RUNNING; // новое состояние

			SetServiceStatus(hServiceStatus, &service_status);// изменяем текущее
состояние службы

			DWORD dwExitCode = SvcMain(dwArgc, lpszArgv);

			if (dwExitCode != 0) // возвращаем код ошибки
			{
				service_status.dwWin32ExitCode = ERROR_SERVICE_SPECIFIC_ERROR;
				service_status.dwServiceSpecificExitCode = dwExitCode;
			}
			else
			{
				service_status.dwWin32ExitCode = NO_ERROR;
			}
		}

		service_status.dwCurrentState = SERVICE_STOPPED; // новое состояние
		SetServiceStatus(hServiceStatus, &service_status);// задаем конечное состояние
службы
	}
}

/*-------------Точка входа в приложение-----------------*/
int _tmain(int argc, LPTSTR argv[])
{
	_tsetlocale(LC_ALL, TEXT(""));

	if (argc < 2)
	{
		_tprintf(TEXT("> Не указан параметр.\n"));
		return 0; // завершаем работу приложения
	} // if


	if (_tcscmp(argv[1], TEXT("/runservice")) == 0) // начало работы службы
	{
		//  инициализируем структуру сервисов
		SERVICE_TABLE_ENTRY service_table[] =
		{
			{(LPTSTR)service_name, ServiceMain},// имя сервиса и функция сервиса
			{NULL, NULL} // больше сервисов нет
		};
		if (!StartServiceCtrlDispatcher(service_table))
		{
			out.open("C:\\ServiceFile.log");
			out << "Start service control dispatcher failed.";
			out.close();

			return 0;
		}
	}

	if (_tcscmp(argv[1], TEXT("/create")) == 0) // создание службы
	{
			// связываемся с менеджером сервисов
			SC_HANDLE hServiceControlManager = OpenSCManager
			(NULL, //локальная машина
				NULL, //активная база данных сервисов
				SC_MANAGER_CREATE_SERVICE //возможно создание сервиса
			);
```

```
                    if (hServiceControlManager == NULL)
                    {
                           std::cout << "Open service control manager failed." << std::endl
                                  << "The last error code: " << GetLastError() << std::endl
                                  << "Press any key to exit." << std::endl;
                           std::cin.get();

                           return -1;
                    }


                    TCHAR CmdLine[MAX_PATH + 13]; // командная строка

                    // определяем путь и имя исполняемого файла
                    GetModuleFileName(NULL, CmdLine, _countof(CmdLine));
                    // добавляем аргумент командной строки
                    StringCchCat(CmdLine, _countof(CmdLine), TEXT(" /runservice"));

                    // создаем службу
                    SC_HANDLE hService = CreateService(
                           hServiceControlManager, //дескриптор менеджера сервисов
                           service_name, //внтуренее имя сервиса
                           SvcDisplayName, //внешнее имя отображаемое
                           0,//нет полных прав контроля сервиса
                           SERVICE_WIN32_OWN_PROCESS, // сервис является процессом
                           SERVICE_DEMAND_START, // служба запускается "вручную"
                           SERVICE_ERROR_NORMAL, CmdLine, NULL, NULL, NULL, NULL, NULL);

                    if (hService == NULL)
                    {
                           std::cout << "Create service failed." << std::endl
                                  << "The last error code: " << GetLastError() << std::endl
                                  << "Press any key to exit." << std::endl;
                           std::cin.get();

                           // закрываем дескриптор менеджера сервисов
                           CloseServiceHandle(hServiceControlManager);

                           return 0;
                    }
                    else
                    {
                           std::cout << "Service is installed." << std::endl;

                           std::cin.get();

                           // закрываем дескрипторы
                           CloseServiceHandle(hService);
                           CloseServiceHandle(hServiceControlManager);

                           return 0;
                    }
             }

      if (_tcscmp(argv[1], TEXT("/delete")) == 0) // удаление службы
      {
             // открываем SCM
             SC_HANDLE hServiceControlManager = OpenSCManager(NULL, NULL,
SC_MANAGER_CREATE_SERVICE);//третий параметр - соединение с менеджером


             if (hServiceControlManager == NULL)
             {
                    std::cout << "Open service control manager failed." << std::endl
                           << "The last error code: " << GetLastError() << std::endl
                           << "Press any key to continue." << std::endl;
                    std::cin.get();
```

```
		return -1;
	}
	else
	{
		std::cout << "Service is opened." << std::endl;
		std::cin.get();
	}


	// открываем службу для удаления
	SC_HANDLE hService = OpenService(hServiceControlManager, service_name, DELETE);

	if (hService == NULL)
	{
		std::cout << "Open service failed." << std::endl
			<< "The last error code: " << GetLastError() << std::endl
			<< "Press any key to exit." << std::endl;
		std::cin.get();

		// закрываем дескриптор менеджера сервисов
		CloseServiceHandle(hServiceControlManager);
		return 0;
	}

	// получаем состояние сервиса
	if (!QueryServiceStatus(hService, &service_status))
	{
		std::cout << "Query service status failed." << std::endl
			<< "The last error code: " << GetLastError() << std::endl
			<< "Press any key to exit." << std::endl;
		std::cin.get();

		// закрываем дескрипторы
		CloseServiceHandle(hServiceControlManager);
		CloseServiceHandle(hService);

		return 0;
	}

	// если сервис работает, то останавливаем его
	if (service_status.dwCurrentState != SERVICE_STOPPED)
	{
		std::cout << "Service is working. It will be stoped" << std::endl;
		if (!ControlService(hService, SERVICE_CONTROL_STOP, &service_status))
		{
			std::cout << "Control service failed." << std::endl
				<< "The last error code: " << GetLastError() << std::endl
				<< "Press any key to exit." << std::endl;
			std::cin.get();

			// закрываем дескрипторы
			CloseServiceHandle(hServiceControlManager);
			CloseServiceHandle(hService);

			return 0;
		}
		// ждем, пока сервис остановится
		Sleep(500);
	}

	if (!DeleteService(hService))
	{
		std::cout << "Delete service failed." << std::endl
			<< "The last error code: " << GetLastError() << std::endl
			<< "Press any key to exit." << std::endl;
		std::cin.get();
		// закрываем дескрипторы
```

```
                        CloseServiceHandle(hServiceControlManager);
                        CloseServiceHandle(hService);

                        return 0;
                }
                else
                {
                        std::cout << "The service is deleted." << std::endl
                                << "Press any key to exit." << std::endl;
                        std::cin.get();
                }
                CloseServiceHandle(hService);
                CloseServiceHandle(hServiceControlManager); // закрываем дескриптор
                return 0; // завершаем работу приложения
        }
        _tprintf(TEXT("> Неизвестный параметр.\n"));
}
```

Листинг 3. Код описывающий работу сервера

```
HANDLE hPipe = INVALID_HANDLE_VALUE; // дескриптор канала

HANDLE hStopper = NULL; // событие для завершения работы службы

HANDLE hThreads; // дескрипторы созданных потоков

// функции потока, где обрабатываются запросы из канала
unsigned __stdcall ThreadFuncPipe(void *lpParameter);

// ----------------------------------------------------------------------------------------------
--
BOOL OnSvcInit(DWORD dwArgc, LPTSTR *lpszArgv)
{
        SECURITY_DESCRIPTOR sd; // дескриптор безопасности

        // инициализируем дескриптор безопасности
        BOOL RetRes = InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION);

        if (FALSE != RetRes)
                RetRes = SetSecurityDescriptorDacl(&sd, TRUE, NULL, FALSE);

        if (FALSE != RetRes)
        {
                SECURITY_ATTRIBUTES sa = { sizeof(SECURITY_ATTRIBUTES) };
                sa.lpSecurityDescriptor = &sd;

                // /// //

                // создаём канал
                hPipe = CreateNamedPipe(TEXT("\\\\.\\pipe\\test_pipe"),
                        PIPE_ACCESS_DUPLEX | FILE_FLAG_OVERLAPPED, // указываем, что канал доступен
для чтения и записи данных
                        PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT, // указываем режим
работы канала
                        PIPE_UNLIMITED_INSTANCES, 0, 0, 0, &sa);

                if (INVALID_HANDLE_VALUE == hPipe)
                {
                        _tprintf(TEXT("> Не удалось создать канал.\n"));
                        return FALSE;
                }

        }

        return RetRes;
}
```

```
void OnSvcStop(void)
{
	// завершаем работу  потока
	SetEvent(hStopper);
}
// ------------------------------------------------------------------------------------------------
--
DWORD SvcMain(DWORD dwArgc, LPTSTR *lpszArgv)
{
	// создаём событие для завершения работы службы
	hStopper = CreateEventEx(NULL, NULL, CREATE_EVENT_MANUAL_RESET, EVENT_ALL_ACCESS);

	// создаём поток, в котором будем обрабатывать запросы из канала
	hThreads = (HANDLE)_beginthreadex(NULL, 0, ThreadFuncPipe, NULL, 0, NULL);
	WaitForSingleObject(hThreads,INFINITE);

	CloseHandle(hStopper); // закрываем дескриптор события

	return 0;
}

// ------------------------------------------------------------------------------------------------
--

// список студентов
constexpr LPCTSTR Students[] =
{
	L"Абаньшин Виктор Андреевич",
	L"Агафонов Данил Сергеевич",
	L"Анистратов Дмитрий Владимирович",
	L"Анистратов Евгений Владимирович",
	L"Бендриков Александр Сергеевич",
	L"Богунов Артем Александрович",
	L"Курбатова Софья Андреевна",
	L"Мануков Давид Альбертович"
};

#pragma pack(push, 1)

struct REQUEST
{
	DWORD PID;
	DWORD index;
}; // struct REQUEST

#pragma pack(pop)
unsigned __stdcall ThreadFuncPipe(void *lpParameter)
{
	// создаём событие для ассинхронных операций с каналом
	HANDLE hPipeEvent = CreateEventEx(NULL, NULL, 0, EVENT_ALL_ACCESS);

	// массив событий
	HANDLE hEvents[2] = { hStopper, hPipeEvent };

	for (;;)
	{
		// инициализируем структуру OVERLAPPED ...
		OVERLAPPED oConnect = { 0 };

		oConnect.hEvent = hPipeEvent;

		ConnectNamedPipe(hPipe, &oConnect);// ожидаем процесс-клиент и образуем с ним
соединение

		DWORD dwResult = WaitForMultipleObjects(2, hEvents, FALSE, INFINITE);// ожидаем одно
из двух событий
```

```
		if ((WAIT_OBJECT_0 == dwResult) || (ERROR_SUCCESS != oConnect.Internal))
		{
			break; // (!) выходим из цикла
		} // if

		for (;;)
		{
			REQUEST Request; // запрос
			DWORD nBytes;
			// чтение данных из канала
			BOOL bRet = ReadFile(hPipe, &Request, sizeof(Request), &nBytes, NULL);
			if (FALSE == bRet) break; // (!) ошибка: выходим из цикла

			TCHAR Response[100] = TEXT(""); // ответ

			if (Request.index < _countof(Students))
			{
				StringCchCopy(Response, _countof(Response), Students[Request.index]);
			} // if

			// запись данных в почтовый канал
			WriteFile(hPipe, Response, sizeof(Response), &nBytes, NULL);
		}

		DisconnectNamedPipe(hPipe); // разрываем соединение
	}

	// закрываем дескриптор события
	CloseHandle(hPipeEvent);

	// закрываем дескриптор канала
	CloseHandle(hPipe);

	return 0;
}
```

Листинг 4. Код описывающий работу клиента

```
#pragma pack(push, 1)
struct REQUEST
{
	DWORD PID;
	DWORD index;
};
#pragma pack(pop)

REQUEST Request; // запрос
DWORD nBytes;
TCHAR Response[100]; // ответ

TCHAR data[8][255] = { L"" };// список полученных данных

int getData();// для получения данных, вернем сколько получено

// ------------------------------------------------------------------------------------------------
--
int _tmain(int argc, LPCTSTR argv[])
{
	_tsetlocale(LC_ALL, TEXT(""));

	_tprintf(TEXT("> Аргументы командной строки:\n"));
	_tprintf(TEXT("> \n"));

	for (int i = 0; i < argc; ++i)
	{
		_tprintf(TEXT("> %s\n"), argv[i]);
```

```
	}

	_tprintf(TEXT("\n"));


	if (2 == argc)
	{
		if (_tcsicmp(argv[1], TEXT("/get_data_list")) == 0) // именованные каналы
		{
			int count = getData();
			for (int i = 0; i < count; i++)
			{
				int num = i + 1;
				_tprintf(TEXT(">%d %s\n"),num , data[i]);
			}
		}
	}
	if (argc == 3)
	{
		if (_tcsicmp(argv[1], TEXT("/get_by_number")) == 0)
				{
				int count = getData();
					if (argv[2])
					{
						int input = _ttoi(argv[2]) - 1;
						for (int i = 0; i < count; i++)
						{
							if (input == i)
							{
								_tprintf(TEXT("> %s\n"), data[i]);
							}
						}
					}
				}
	}


}

int getData()
{
	int count = 0;
	for (;;)
	{
		// соединение с именованным каналом
		//pipe - это  менять нельзя.
	BOOL RetRes = CallNamedPipe(TEXT("\\\\.\\pipe\\test_pipe"),
			&Request, sizeof(Request), Response, sizeof(Response), &nBytes,
NMPWAIT_WAIT_FOREVER);

		if (RetRes != FALSE)
		{
			if (_T('\0') == Response[0])
			{
				return count;
			}
			StringCchCat(data[count++], _countof(data[count++]), Response);//заполним
список данными полученными от сервера
			Sleep(200);
		}
		else
		{
			_tprintf(TEXT("> Ошибка: %d\n"), GetLastError());
			return count;
		}
		++Request.index;
	}
```

```
}
```

3. Тестирование:

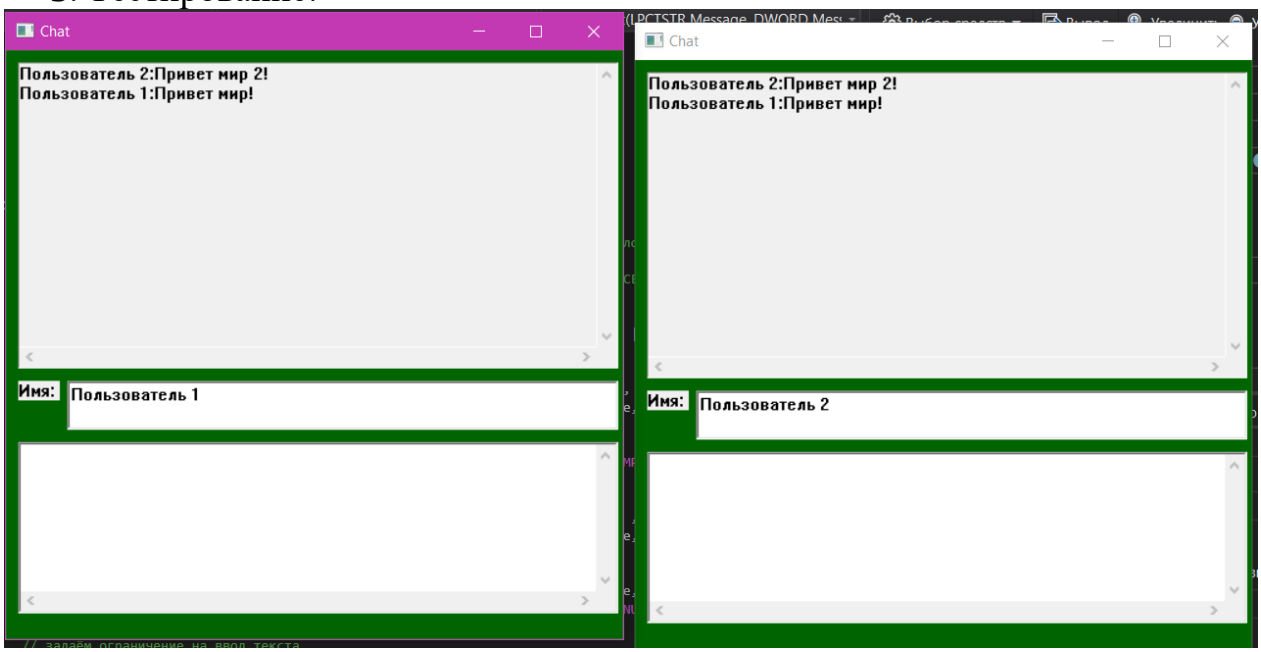

Рис. 6.2. Приложение 1

```
C:\Users\Kurbatova\source\LW2020\lw_2020\lw_os_6\lw_os_6\Debug>server.exe /delete
Service is opened.

The service is deleted.
Press any key to exit.


C:\Users\Kurbatova\source\LW2020\lw_2020\lw_os_6\lw_os_6\Debug>
```

Рис. 6.3. Удаление созданной службы

```
C:\Users\Kurbatova\source\LW2020\lw_2020\lw_os_6\lw_os_6\Debug>server.exe /create
Service is installed.


C:\Users\Kurbatova\source\LW2020\lw_2020\lw_os_6\lw_os_6\Debug>
```

Рис. 6.4. Создание службы

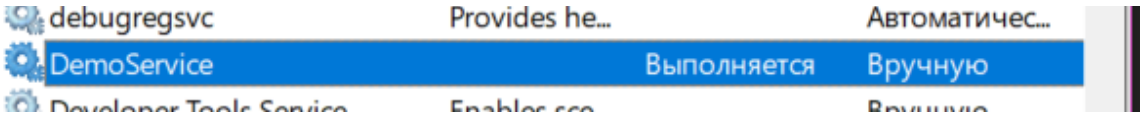

Рис. 6.5. запуск службы в диспетчере служб

```
C:\Users\Kurbatova\source\LW2020\lw_2020\lw_os_6\lw_os_6\Debug>client.exe /get_data_list
> Аргументы командной строки:
>
> client.exe
> /get_data_list

>1 Абаньшин Виктор Андреевич
>2 Агафонов Данил Сергеевич
>3 Анистратов Дмитрий Владимирович
>4 Анистратов Евгений Владимирович
>5 Бендриков Александр Сергеевич
>6 Богунов Артем Александрович
>7 Курбатова Софья Андреевна
>8 Мануков Давид Альбертович
```

Рис. 6.6. выполнение запроса к службе: вывод всего списка

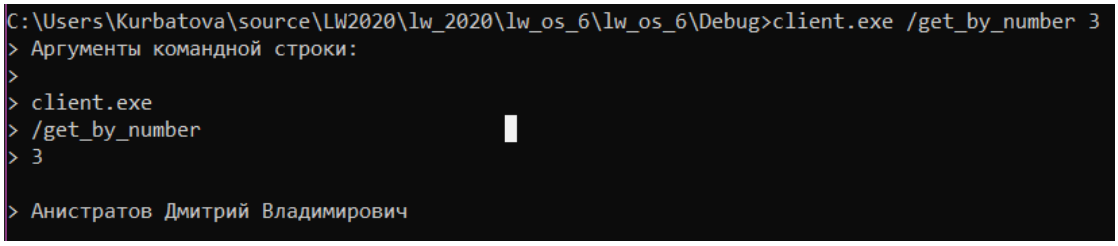

Рис. 6.7. выполнение запроса к службе: получение элемента по его номеру

Вывод: Таким образом, в ходе выполнения лабораторной работы было разработано 2 приложения, которые позволили изучить межпроцессорное взаимодействие в Windows.