**Помечайте исправленные задания пж**

# Практика

## Задание 1

1. **Создание и отладка файла DLL. Для демонстрации работы написать две элементарные функции в библиотеке и вызвать их из основной программы**

**В проекте DLL:**

**Создаем проект и в свойствах проекта в “Свойства конфигурации - общее” меняем “Тип конфигурации” на Динамическую библиотеку. Далее в “С/С++ - Дополнительно” ставим компилировать как Си. Затем переходим в свойствах в “Компоновщик - Все параметры” ставим в “Подсистеме” Windows**

**В этом проекте:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

BOOL WINAPI DllMain(HINSTANCE hlnstDll, DWORD dwReason, LPVOID IpReserved)

{

BOOL bAllWentWell = TRUE;

switch (dwReason)

{

case DLL\_PROCESS\_ATTACH:

break;

case DLL\_THREAD\_ATTACH:

break;

case DLL\_THREAD\_DETACH:

break;

case DLL\_PROCESS\_DETACH:

break;

}

if (bAllWentWell)

return TRUE;

else

return FALSE;

}

extern \_\_declspec(dllimport) int Sum(int a, int b);

extern \_\_declspec(dllimport) int Razn(int a, int b);

int Sum(int a, int b)

{

return a + b;

}

int Razn(int a, int b)

{

return a - b;

}

**Если доебется и попросит компилировать как С++ поставить, то в строчке с импортом функции нужно будет добавить “C” после слова extern:**

extern "C" \_\_declspec(dllimport) int Sum(int a, int b);

extern "C" \_\_declspec(dllimport) int Razn(int a, int b);

int Sum(int a, int b)

{

return a + b;

}

int Razn(int a, int b)

{

return a - b;

}

**Далее создаем основной проект.**

**В свойствах проекта в “С/С++ - Дополнительно” ставим компилировать как Си. Затем переходим в свойствах в “Компоновщик - Все параметры” ставим в “Подсистеме” Windows если вывод ответа в консоль не требуется, если нужно, то просто оставить Консоль. Если без демонстрации в консоль, то для показа результата нужно поставить точку останова на строке *FreeLibrary(hMyDLL);***

#include <Windows.h>

#define PATH L"DLLCode.dll" (путь к ДЛЛ лежит в папке решения в Debug”

typedef int(\_cdecl\* MyFunction)(int, int);

int main(void)

{

system("chcp 1251>nul");

HINSTANCE hMyDLL;

if ((hMyDLL = LoadLibrary(PATH)) == NULL) return 1;

MyFunction myFuncS = (MyFunction)GetProcAddress(hMyDLL, "Sum");

int s = myFuncS(6, 3);

MyFunction myFuncR = (MyFunction)GetProcAddress(hMyDLL, "Razn");

int r = myFuncR(6,3);

FreeLibrary(hMyDLL);

return 0;

}

## Задание 2

1. **Программа для считывания и записи строкового значения с использованием текстового файла. Для работы с файлами используются функции WinAPI**

#define \_CRT\_SECURE\_NO\_WARNINGS//макрос позволяет игнорировать предупреждение об использовании небезопасных функций

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void main()

{

system("chcp 1251>nul");//локализация на русский

FILE\* Chtenie;//файловый тип (структура), здесь содаем переменную для чтения

FILE\* Zapis;//файловый тип (структура), здесь содаем переменную для записи

Zapis = fopen("C:\\Users\\eliza\\Desktop\\Doc.txt", "w");//открываем файл по его имени ("file.txt") и задаем параметр на создание нового файла ("w") (об остальных типах ниже)

fprintf(Zapis, "Строка для записи в файлы");//fprintf записывает в файл строку: Zapis - структура с названием файла, "Строка для записи в файлы" - угадай, че

fclose(Zapis);//закрываем работу с файлом на запись, так как иначе его нельзя будет прочитать

char Stroka[30];//массив char символов (строка) для чтения из файла, пока пустой

Chtenie = fopen("C:\\Users\\eliza\\Desktop\\Doc.txt", "r");//также открываем файл по имени (то же имя!) и задаем параметр "r" (чтение)

int k = 0;//переменная счетчик прочитанных символов

for (k = 0; !feof(Chtenie); k++)//запускаем цикл, который будет перебирать все символы в файле, пока файл не закончится

{

fscanf(Chtenie, "%c", &Stroka[k]);//читаем по одному символу и записываем его в строку

}

Stroka[k - 1] = '\0';//добавляем к строке закрывающий 0, чтобы не записать лишнего из памяти (почему - 1, я не знаю, иначен он в конце добавляет букву М)

printf(Stroka);//выводим всю строку в консоль

fclose(Chtenie);//закрываем файл

}

## //Задано Задание 3

1. **Получение названия клавиши с использованием функции GetKeyNameText (Windows HOOK). Название клавиши выводить в MessageBox**

#include <Windows.h>

#include <stdio.h>

#define PATH L"Log.txt"

#define RUS 1049

#define ENG 1033

#define SIZE\_STR 20

BOOL IsCaps(void);

LRESULT CALLBACK LogKey(int iCode, WPARAM wParam, LPARAM lParam);

VOID WriteToFile(LPWSTR wstr);

WCHAR EnToRus(WCHAR c);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE HpREViNSTANCE, PSTR pCmdLine, int nCmdShow)

{

HHOOK hHook = SetWindowsHookExW(WH\_KEYBOARD\_LL, LogKey, NULL, NULL);

MSG msg = { 0 };

while (GetMessageW(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hHook);

return 0;

}

BOOL IsCaps(void)

{

if ((GetKeyState(VK\_CAPITAL) & 0x0001) != 0 ^ (GetKeyState(VK\_SHIFT) & 0x8000) != 0)

return TRUE;

return FALSE;

}

LRESULT CALLBACK LogKey(int iCode, WPARAM wParam, LPARAM lParam)

{

if (wParam == WM\_KEYDOWN)

{

PKBDLLHOOKSTRUCT pHook = (PKBDLLHOOKSTRUCT)lParam;

WORD KeyLayout = LOWORD(GetKeyboardLayout(GetWindowThreadProcessId(GetForegroundWindow(), 0)));

DWORD iKey = MapVirtualKeyW(pHook->vkCode, NULL) << 16;

if (!(pHook->vkCode <= 1 << 5))

iKey |= 0x1 << 24;

LPWSTR KeyString = (LPWSTR)calloc(SIZE\_STR + 1, sizeof(WCHAR));

GetKeyNameTextW(iKey, KeyString, SIZE\_STR);

if (wcslen(KeyString) > 1)

{

WriteToFile(L"[");

WriteToFile(KeyString);

WriteToFile(L"]");

}

else

{

if (!IsCaps()) KeyString[0] = tolower(KeyString[0]);

if (KeyLayout == ENG)

{

}

if (KeyLayout == RUS)

{

KeyString[0] = EnToRus(KeyString[0]);

}

WriteToFile(KeyString);

}

free(KeyString);

}

return CallNextHookEx(NULL, iCode, wParam, lParam);

}

VOID WriteToFile(LPWSTR wstr)

{

FILE\* f = NULL;

if (!\_wfopen\_s(&f, PATH, L"ab"))

{

fwrite(wstr, sizeof(WCHAR), wcslen(wstr), f);

fclose(f);

}

}

WCHAR EnToRus(WCHAR c)

{

switch (c)

{

case L'q': return L'й';

case (L'w'): return L'ц';

case (L'e'):return L'у';

case (L'r'):return L'к';

case (L't'): return L'е';

case (L'y'): return L'н';

case (L'u'): return L'г';

case (L'i'): return L'ш'; break;

case (L'o'): return L'щ'; break;

case (L'p'): return L'з'; break;

case (L'['): return L'х'; break;

case (L']'): return L'ъ'; break;

case (L's'): return L'ы'; break;

case (L'd'): return L'в'; break;

case (L'f'): return L'а'; break;

case (L'g'): return L'п'; break;

case (L'h'): return L'р'; break;

case (L'j'): return L'о'; break;

case (L'k'): return L'л'; break;

case (L'l'): return L'д'; break;

case (L';'): return L'ж'; break;

case (L'\''): return L'э'; break;

case (L'z'): return L'я'; break;

case (L'x'): return L'ч'; break;

case (L'c'): return L'с'; break;

case (L'v'): return L'м'; break;

case (L'b'): return L'и'; break;

case (L'n'): return L'т'; break;

case (L'm'): return L'ь'; break;

case (L','): return L'б'; break;

case (L'/'): return L'.'; break;

case (L'`'): return L'ё'; break;

case (L'\\'): return L'\\'; break;

case (L'#'): return L'№'; break;

case (L'$'): return L'; '; break;

case (L'^'): return L':'; break;

case (L'&'): return L'?'; break;

case (L'|'): return L'/'; break;

case (L'Q'): return L'Й'; break;

case (L'W'): return L'Ц'; break;

case (L'E'): return L'У'; break;

case (L'R'): return L'К'; break;

case (L'T'): return L'Е'; break;

case (L'Y'): return L'Н'; break;

case (L'U'): return L'Г'; break;

case (L'I'): return L'Ш'; break;

case (L'O'): return L'Щ'; break;

case (L'P'): return L'З'; break;

case (L'{'): return L'Х'; break;

case (L'}'): return L'Ъ'; break;

case (L'A'): return L'Ф'; break;

case (L'S'): return L'Ы'; break;

case (L'D'): return L'В'; break;

case (L'F'): return L'А'; break;

case (L'G'): return L'П'; break;

case (L'H'): return L'Р'; break;

case (L'J'): return L'О'; break;

case (L'K'): return L'Л'; break;

case (L'L'): return L'Д'; break;

case (L':'): return L'Ж'; break;

case (L'"'): return L'Э'; break;

case (L'Z'): return L'Я'; break;

case (L'X'): return L'Ч'; break;

case (L'C'): return L'С'; break;

case (L'V'): return L'М'; break;

case (L'B'): return L'И'; break;

case (L'N'): return L'Т'; break;

case (L'M'): return L'Ь'; break;

case (L'<'): return L'Б'; break;

case (L'>'): return L'Ю'; break;

default: return L"Ъ";

}

}

## Задание 4

1. **Получение кода клавиши в формате ASCII и использованием Windows HOOK. Название клавиши выводить в MessageBox**

#include <Windows.h>

#include <stdio.h>

#define RUS 1049//страница русского языка

#define ENG 1033//страница английского языка

#define SIZE\_STR 20//длина строки для записи

BOOL IsCaps(void);//прототип функции проверки на капс (без прототипа функция не будет работать, так как она объявлена ниже мейна)

LRESULT CALLBACK LogKey(int iCode, WPARAM wParam, LPARAM lParam);//прототип основной функции хука, перехват нажатия на клавиши

WCHAR EnToRus(WCHAR c);//прототип функции перевода английских символов в русские

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR pCmdLine, int nCmdShow)

{

HHOOK hHook = SetWindowsHookExW(WH\_KEYBOARD\_LL, LogKey, NULL, NULL);//дескриптор хука. Про SetWindowsHookExW подробнее ниже

MSG msg = { 0 };//структура системного сообщение. Нажатие на клавишу, по сути, и есть системное сообщение, которое нам надо получить, но пока ноль

while (GetMessageW(&msg, NULL, 0, 0))//GetMessageW позволяет получить сис сообщение (присвоем его &msg, остальные аргументы - по умолчанию)

{ //До тех пор, пока мы будем получать сообщения, они будут переводится

TranslateMessage(&msg);//TranslateMessage переводит сис сообщение (&msg) в символы или код, как нам понадобится

DispatchMessage(&msg);//DispatchMessage выделяет сообщение, чтобы его можно было использовать

}

UnhookWindowsHookEx(hHook);//после завершения наших дел снимаем хук, чтобы он не нагружал систему

return 0;//если все выполнено без ошибок, завершаем прогу с кодом 0

}

LRESULT CALLBACK LogKey(int iCode, WPARAM wParam, LPARAM lParam)//осноная функция хука

{

if (wParam == WM\_KEYDOWN)//получаем сис сообщение, и если оно говорит о нажатии на клавишу, делаем дальнейший код

{

PKBDLLHOOKSTRUCT pHook = (PKBDLLHOOKSTRUCT)lParam;//струтура хука для определения клавиш

WORD KeyLayout = LOWORD(GetKeyboardLayout(GetWindowThreadProcessId(GetForegroundWindow(), 0)));//определяем раскладку по номеру потока этой проги

LPWSTR KeyString = (LPWSTR)calloc(SIZE\_STR + 1, sizeof(WCHAR));//строка для записи названия кода клавиши, пока выделяем память (20 байт + закрывающий 0 \* на 2 байта)

swprintf(KeyString, 512, L"%d", pHook->vkCode);//перевод кода клавиши из ASCII в строку: KeyString - строка для записи, 512 - объем памяти,

//L"%d" - тип данных, из которого переводим, pHook->vkCode - код клавиши

MessageBox(NULL, KeyString, L"Буква", MB\_OK);//выводим полученный код в месседжбоксе

free(KeyString);//освобождаем память под строку, чтобы она не занимала ее

}

return CallNextHookEx(NULL, iCode, wParam, lParam);//завершаем эту функцию, с помощью CallNextHookEx вызываем следующие хуки, которые продолжат перехват нажатий

}

//SetWindowsHookExW устанавливает хук на определенные действия и задает функцию, которая должна вызываться при этом действии

//Аргументы: WH\_KEYBOARD\_LL - устанавливает хук на клаву, LogKey - функция, которая будет вызываться , NULL - какой-то режим (NULL по умолчанию), NULL - поток (NULL - нет его))

## Задание 5

1. **Программа для преобразования числа в строку и склейки двух строк. Разработать без использования системных функций**

Склеивание:

#include <string.h>

#include <malloc.h>

#include <stdio.h>

int main()

{

system("chcp 1251>nul");

char\* str1 = "Первая строка";

char\* str2 = ", а вот и вторая строка";

char\* srtResult = calloc(strlen(str1) + strlen(str2)-1, sizeof(char));

int lenstr = 0;

for (size\_t i = 0; i < strlen(str1); i++)

{

srtResult[lenstr] = str1[i];

lenstr++;

}

for (size\_t i = 0; i < strlen(str2); i++)

{

srtResult[lenstr] = str2[i];

lenstr++;

}

srtResult[lenstr] = '\0';

printf("%s", srtResult);

}

## Задание 6

1. **Программа с использованием нескольких программных модулей (каждый модуль должен располагаться в отдельном файле .c). В рамках модулей должна быть как минимум одна пользовательская функция и одна структура. Также для этих модулей должен быть написан заголовочный файл, содержащий всю необходимую информацию для корректного функционирования модулей**

**Основной код:**

#include "Header.h"

int main()

{

struct s\* us = calloc(1, sizeof(struct s));

us->a = 10;

us->b = 2;

printf("%d\n", sum(us));

}

**Заголовочный:**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <malloc.h>

#include <Windows.h>

#include "Struct.c"

#include "Func.c"

**Функция:**

sum(struct s\* us)

{

return us.a + us.b;// передаем параметры структуры в переменные

}// надо точки ставить, они потом превратятся в стрелки

**Структура:**

struct s

{

int a;

int b;

};

## Задание 7

1. **Программа для преобразования строки в число. Строка хранится в виде указателя LPWSTR. Реализовать без использования системных функций**

#include <string.h>

#include <malloc.h>ыФ

#include <stdio.h>

#include <Windows.h>

int stepen(int ig, int strlen)

{

int f = 1;

for (size\_t i = 0; i < strlen - ig; i++)

{

f \*= 10;

}

return f;

}

int main()

{

system("chcp 1251>nul");

LPWSTR str = L"123415677";

int d = 0;

for (size\_t i = 0; i < lstrlenW(str); i++)

{

d += (str[i] - 48) \* stepen(i, lstrlenW(str)-1);

}

int g = 0;

}

## Задание 8

1. **Программа для записи строки в системный буфер обмена**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int ClipboardInputText(LPWSTR buffer)

{

DWORD message;

HANDLE hMes; // дескриптор глобальной области памяти

message = wcslen(buffer) + 1; // так как в юникоде в конце строки стоит “/0” надо считтаь на 1 символ больше

hMes = GlobalAlloc(GMEM\_MOVEABLE, message \* sizeof(LPWSTR));// Global - выделяем память в глобальной области видимости, чтобы память была видна за пределами программы

memcpy(GlobalLock(hMes), buffer, message \* sizeof(LPWSTR));

GlobalUnlock(hMes); // делает содержимое доступным

OpenClipboard(0); // открывает буфер обмена

EmptyClipboard(); // очиска буфера обмена

SetClipboardData(CF\_UNICODETEXT, hMes);

CloseClipboard(); // закрыть буфер, сделав его доступным для других приложений

return 0;

}

int main()

{

system("chcp 1251 > null");

LPWSTR Message = L"Текст для записи в буфер";

ClipboardInputText(Message);

}

## Задание 9

1. **Программа для считывания данных из системного буфера обмена**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int ClipboardOutptText()

{

LPWSTR Mess = NULL;

OpenClipboard(NULL); // открыть буфер обмена - код NULL в режиме по умолчанию

HANDLE hClipboardData = GetClipboardData(CF\_UNICODETEXT);

Mess = (LPWSTR)GlobalLock(hClipboardData); // считать из глобального участка памяти, привести это в к строке (LPWSTR) и поместить в строку Messege

GlobalUnlock(hClipboardData);

CloseClipboard();// делаем буфер доступным для других приложений

MessageBox(NULL, Mess, L"Содержимое буфера", MB\_OK);

return 0;

}

int main()

{

system("chcp 1251 > null");

ClipboardOutptText();

}

## Задание 10

1. **Создание программы с критической секцией. Программа должна содержать минимум два потока. Использование критической секции в функции потока должно быть обосновано.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <malloc.h>

#define I\_WILL\_WAIT 10

CRITICAL\_SECTION section = { 0 }; //Критическая секция

VOID Crit(VOID);

DWORD WINAPI Thread1(DWORD param);

DWORD WINAPI Thread2(DWORD param);

int count = 0;

HANDLE h[3];

int main(void)

{

Crit();

return 0;

}

VOID Crit(VOID)

{

DWORD tmp1 = 1;

DWORD tmp2 = 2;

DWORD tmp3 = 3;

InitializeCriticalSection(&section);

h[0] = CreateThread(NULL, 0, Thread2, tmp1, 0, 0);

h[1] = CreateThread(NULL, 0, Thread2, tmp2, 0, 0);

h[2] = CreateThread(NULL, 0, Thread2, tmp3, 0, 0);

WaitForMultipleObjects(3, h, TRUE, INFINITE);

DeleteCriticalSection(&section);

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

}

DWORD WINAPI Thread1(DWORD param)

{

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

{

Sleep(I\_WILL\_WAIT);

count++;

printf("count = %d, potok = %d\n", count, param);

}

ExitThread(0);

}

DWORD WINAPI Thread2(DWORD param)

{

EnterCriticalSection(&section);

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

{

Sleep(I\_WILL\_WAIT);

count++;

printf("count = %d, potok = %d\n", count, param);

}

LeaveCriticalSection(&section);

ExitThread(0);

}

## Задание 11

1. **Программа для считывания нескольких числовых значений из строки и из записи в числовой массив (разделитель - пробел). Количество значений в строке заранее неизвестно.**

#include <stdio.h>

#include <Windows.h>

int pow(int a, int b) //

{

for (size\_t i = 0; i < b; i++)

{

a \*= 10;

}

return a;

}

int main()

{

LPSTR str = "10 52 654";

int index = 0;

int count = 0;

int countall = 0;

while (str[index] != L'\0')

{

if (str[index] == L' ')

{

count++;

}

countall++;

index++;

}

index = 0;

int\* mass = calloc(count + 1, sizeof(int));

int\* countt = calloc(countall, sizeof(int));

int\* min = calloc(count + 1, sizeof(int));

int ind = 0;

for (int i = 0; i < count + 1; i++)

{

countt[i] = 0;

}

while (str[index] != L'\0')

{

while (str[index] != L' ')

{

if (str[index] == L'\0')

{

break;

}

countt[ind] += 1;

if (str[index] == L'-')

{

min[ind] = 1;

}

index++;

}

ind++;

index++;

}

index = 0;

int ii = 0;

int iii = 0;

int pr = 0;

for (int i = 0; i < count + 1; i++)

{

if (str[index] != L' ')

{

iii = countt[i];

while (countt[i] > ii)

{

if (str[index] != L'-')

{

pr = (int)(str[index] - '0');

mass[i] += pow(pr, iii) / 10;

ii++;

index++;

iii--;

}

else

{

ii++;

index++;

iii--;

}

}

ii = 0;

}

else

{

i--;

}

index++;

}

for (size\_t i = 0; i < count + 1; i++)

{

if (min[i] == 1)

{

mass[i] = -mass[i];

}

}

for (int i = 0; i < count + 1; i++)

{

printf("%d\n", mass[i]);

}

}

## Задание 12

1. **Запись и считывание строковых параметров в системный реестр.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <strsafe.h>

#include <stdio.h>

#include <time.h >

#include <windows.h>

int WINAPI wWinMain(HINSTANCE h, HINSTANCE hh, LPSTR hhh, int hhhh)

{

HKEY hKey = NULL;//дескриттор ключа реестра (он является структурой)

if (RegOpenKey(HKEY\_CURRENT\_USER, NULL, &hKey) != ERROR\_SUCCESS) //открываем раздел HKEY\_CURRENT\_USER

return 1;

if (RegSetValueW(hKey, L"Mykey", REG\_SZ, L"Значение по умолчанию", 22 \* sizeof(WCHAR)) == ERROR\_SUCCESS)

{

MessageBox(NULL, L"Ключ успешно создан и ему присвоено значение по умолчанию", L"Информация", MB\_OK);

}

LPSTR st = L"Привет, привет, ты не спишь";

TCHAR buff[512];

wsprintf(buff, L"%s", st);

if (RegSetValueExW(hKey, L"Mykeys", NULL, REG\_SZ, &buff, 22 \* sizeof(WCHAR)) == ERROR\_SUCCESS)

{

MessageBox(NULL, &buff, L"Строковый параметр", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(NULL, L"Что-то пошло не так", L"Информация", MB\_OK);

}

if (RegDeleteValue(hKey, L"Mykeys") == ERROR\_SUCCESS)

{

MessageBox(NULL, L"Параметр успешно удален", L"Информация", MB\_OK);

}

if (RegDeleteKey(hKey, L"MyKey") == ERROR\_SUCCESS)

{

MessageBox(NULL, L"Ключ успешно удален", L"Информация", MB\_OK);

}

RegCloseKey(hKey);

}

## //Задано Задание 13

1. **Обработка нажатия клавиши мыши в системе (выписать в messagebox какая клавиша нажата и сколько раз)**

#include <Windows.h>

#include <stdio.h>

LRESULT CALLBACK LogMouse(int iCode, WPARAM wParam, LPARAM lParam);

DWORD LKey = 0, RKey = 0, CKey = 0, TKey = 0;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE HpREViNSTANCE, PSTR pCmdLine, int nCmdShow)

{

HHOOK hHook = SetWindowsHookExW(WH\_MOUSE\_LL, LogMouse, NULL, NULL);

MSG msg = { 0 };

while (GetMessageW(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hHook);

return 0;

}

LRESULT CALLBACK LogMouse(int iCode, WPARAM wParam, LPARAM lParam)

{

if (wParam == 513)

{

LKey++;

LPWSTR String = (LPWSTR)calloc(150, sizeof(WCHAR));

swprintf(String, 150, L"Внимание! Лимит нажатий на левую кнопку мыши исчерпан. Вы нажали %d раз", LKey);

MessageBox(NULL, String, L"Зафиксировано несанкционированное нажатие кнопки мыши", MB\_OK);

free(String);

}

if (wParam == 516)

{

RKey++;

LPWSTR String = (LPWSTR)calloc(150, sizeof(WCHAR));

swprintf(String, 150, L"Внимание! Лимит нажатий на правую кнопку мыши исчерпан. Вы нажали %d раз", RKey);

MessageBox(NULL, String, L"Зафиксировано несанкционированное нажатие кнопки мыши", MB\_OK);

free(String);

}

if (wParam == 520)

{

CKey++;

LPWSTR String = (LPWSTR)calloc(150, sizeof(WCHAR));

swprintf(String, 150, L"Внимание! Лимит нажатия на колёсико мыши исчерпан. Вы нажали %d раз", CKey);

MessageBox(NULL, String, L"Зафиксировано несанкционированное нажатие кнопки мыши", MB\_OK);

free(String);

}

//вверх и вниз считает вместе

if (wParam == 522)

{

TKey++;

LPWSTR String = (LPWSTR)calloc(150, sizeof(WCHAR));

swprintf(String, 150, L"Внимание! Лимит прокрутки колёсика мыши исчерпан. Вы прокрутил %d раз", TKey);

MessageBox(NULL, String, L"Зафиксировано несанкционированное нажатие кнопки мыши", MB\_OK);

free(String);

}

return CallNextHookEx(NULL, iCode, wParam, lParam);

}

## Задание 14

1. **Использование системного буфера обмена для передачи строковых значений между процессами**

**Основной код:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <strsafe.h>

#include <Windows.h>

#include < stdlib.h >

LPWSTR ClipboardOutputText();

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR pCmdLine, int nCmdShow)

{

int p = 1;

while (p)

{

ClipboardInputText(L"HeLLO"); //

Sleep(1000);

STARTUPINFO si;

PROCESS\_INFORMATION pi = { 0 };

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi)); // все что сверху это для создания процесса

if (!CreateProcessA(NULL, "C:\\Users\\Dima\\source\\repos\\Project8\\x64\\Debug\\Project1.exe", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))// проверка на запуск процесса

{

}

WaitForSingleObject(pi.hProcess, INFINITE);

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

p = 0;

}

return 0;

}

int ClipboardInputText(LPWSTR buffer)// метод для записи строки в буфер обмена

{

DWORD len;

HANDLE hMem;

len = wcslen(buffer) + 1;

hMem = GlobalAlloc(GMEM\_MOVEABLE, len \* sizeof(LPWSTR));

memcpy(GlobalLock(hMem), buffer, len \* sizeof(LPWSTR));

GlobalUnlock(hMem);

OpenClipboard(0);

EmptyClipboard();

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

return 0;

}

**Получение и вывод в процессе**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <strsafe.h>

#include <Windows.h>

#include < stdlib.h >

LPWSTR ClipboardOutputText();

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR pCmdLine, int nCmdShow)

{

int p = 1;

while (p)

{

LPSTR\* Data = ClipboardOutputText();

MessageBoxW(NULL, Data, NULL, MB\_OK | MB\_ICONWARNING);

Sleep(1000);

p = 0;

}

return 0;

}

TCHAR\* ClipboardOutputText()// метод для получения данных из буфера обмена

{

TCHAR\* Mess = NULL;

OpenClipboard(NULL); //открыть буфер обмена

HANDLE hClipboardData = GetClipboardData(CF\_UNICODETEXT); //записать в буфер обмена данные соответствующего типа

Mess = (TCHAR\*)GlobalLock(hClipboardData); //считать из глобального участка памяти, привести это все к строке

GlobalUnlock(hClipboardData); //освободить глобальные участки памяти

CloseClipboard(); //закрыть буфер обмена, сделать его доступным для других приложений

EmptyClipboard(); //очистить буфер обмена

return Mess;

}

## Задание 15

1. **Запись и считывание числовых параметров в системный реестр**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <strsafe.h>

#include <stdio.h>

int WINAPI wWinMain(HINSTANCE h, HINSTANCE hh, LPSTR hhh, int hhhh)

{

HKEY hKey = NULL;

if (RegOpenKey(HKEY\_CURRENT\_USER, NULL, &hKey) != ERROR\_SUCCESS)

return 1;

DWORD value = 22; //число, которое записываем

if (RegSetValueEx(hKey, TEXT("ParamSize"), 0, REG\_DWORD, (const BYTE\*)&value, sizeof(value)) == ERROR\_SUCCESS)

{

MessageBox(NULL, L"Записался", L"Size", MB\_OK | MB\_ICONINFORMATION);

}

DWORD DataType = 0;

DWORD DataLen = 100500;

DWORD SIZEE;

TCHAR sZ[100];

if (RegGetValueW(HKEY\_CURRENT\_USER, NULL, L"ParamSIze", RRF\_RT\_DWORD, &DataType, &SIZEE, &DataLen) == ERROR\_SUCCESS)

{

StringCbPrintf(sZ, 100, TEXT("%d"), SIZEE);

MessageBox(NULL, sZ, L"Size", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(NULL, L"Что-то пошло не так", L"Информация", MB\_OK);

}

RegCloseKey(hKey);

return 0;

}

## Задание 16

1. **Разработка программы-секундомера. в одном потоке идет отсчет времени, а в другом осуществляется управление секундомером**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int times(long int t)

{

int sec = t;

int min = 0;

int hours = 0;

while (1)

{

Sleep(1000);

sec++;

if (sec == 60)

{

min++;

sec = 0;

if (min == 60)

{

hours++;

min = 0;

if (hours == 24)

{

hours = 0;

}

}

}

system("cls");

printf\_s("%d:%d:%d\n", hours, min, sec);

}

}

int main()

{

system("chcp 1251 >null");

DWORD choose = -1;

HANDLE hTread[1];

printf("\n1 - секундомер\n0 - Остановить\n");

while (1)

{

scanf\_s("%d", &choose);

switch (choose)

{

case 1:

hTread[0] = CreateThread(NULL, 0, times, 0, NULL, NULL);// запускаем поток с секуномером

break;

case 0:

SuspendThread(hTread[0]);// завершаем

break;

default:

break;

}

}

WaitForSingleObject(hTread, INFINITE);

}

## Задание 17

1. **Разработка программы для вычисления факториала в отдельном потоке. В программе должна быть предусмотрена валидация данных**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int FacKU(int n) {

if (n == 0)// проверка

return 1;

if (n < 1)

return 0;

int a = 1;

for (size\_t i = 1; i <= n; i++)

{

a \*= i;

}

printf("%d\n", a);

return a;

}

main() {

HANDLE hThread;

while (1) {

char str[256];

scanf("%s", str);

int n = atoi(str);

if (n != NULL) {

hThread = CreateThread(NULL, 0, FacKU, n, NULL, NULL);// создание потока

}

}

}

## Задание 18

1. **Создание и настройка проекта WinAPI. Вывод сообщения в messagebox. Текст сообщения считывается из текстового файла в кодировке Юникод**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

int WINAPI WinMain()

{

DWORD d;

HANDLE hFile = CreateFile(L"1.txt",//путь к файлу

GENERIC\_READ | GENERIC\_WRITE,//флаги на открытие как на чтение, так и на запись

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,//совместный доступ только на чтение

NULL,//структура безопасности по умолчанию

OPEN\_ALWAYS,//режим создания файла (открыть, перезаписать и т.п.)

FILE\_ATTRIBUTE\_NORMAL,//атрибуты файла по умолчанию

NULL);//шаблон файла отсутствует

LPCSTR str = calloc(100, 1);

ReadFile(hFile, str, 100, &d, NULL);

MessageBoxA(NULL, str, NULL, MB\_OK);

}

## //Задано Задание 19

1. **Программа с запуском стороннего процесса. Имя запускаемого процесса выбирается с помощью системного диалогового окна.**

## Задание 20

1. **Использование условных блоков для анализа значений, возвращаемых системной функцией**

#include <stdio.h>

#include <Math.h>

#define Check 16

int main()

{

system("chcp 1251>nul");

if (pow(2, 4) == Check)

{

printf("Подсчет правильный");

}

else

{

printf("Подсчет не правильный");

}

}

## Задание 21

1. **Программа с импортом функции из файла DLL. Функция принимает структуру, содержащую координаты двух точек, а возвращает длину отрезка, который образуется этими точками**

**Основной код программы:**

#define PATH L"DLLCode.dll"

typedef double(\_cdecl\* getSize)(struct cord\*);

typedef struct coordinats {

COORD point1;

COORD point2;

}cord;

int main()

{

HINSTANCE hMyDLL;

if ((hMyDLL = LoadLibrary(PATH)) == NULL) return 1;

getSize getSqize = (getSize)GetProcAddress(hMyDLL, "getSize");

cord\* cord = malloc(sizeof(cord));

cord->point1.X = 2;

cord->point1.Y = 1;

cord->point2.X = 6;

cord->point2.Y = 4;

double result = getSqize(cord);

FreeLibrary(hMyDLL);

return 0;

}

**Код длл**

#include <Windows.h>

typedef struct coordinats {

COORD point1;

COORD point2;

}cord;

\_\_declspec(dllimport) double getSize(cord\* cord);

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)

{

switch (fdwReason)

{

case DLL\_PROCESS\_ATTACH:

break;

case DLL\_THREAD\_ATTACH:

break;

case DLL\_THREAD\_DETACH:

break;

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

double getSize(cord\* cord)

{

double d = sqrt((pow(cord->point2.X - cord->point1.X, 2) + pow(cord->point2.Y - cord->point1.Y, 2)), 2);

return d;

}

## //Задано Задание 22

1. **Отправка и получение данных из именованного канала. Данные содержатся в строковой форме. Перед отправкой следующей строки клиенту необходимо дождаться ответа от сервера.**

**Сервер**

#include <stdio.h>

#include <Windows.h>

#define SIZE\_BUFFER 512

int main()

{

system("chcp 1251");

HANDLE hNamedPipe;//объявление дескриптора калала

LPWSTR lpszPipeName = "\\\\.\\pipe\\MyPipe";//переменная, содержащая имя канала

DWORD size\_buffer = SIZE\_BUFFER;//размер буфера для чтения

LPSTR buffer = (CHAR\*)calloc(size\_buffer, sizeof(CHAR));//строковая переменная, в которую будут считаны данные

char message[SIZE\_BUFFER];

BOOL Connected;

DWORD actual\_readen; //сколько на самом деле было прочитано байт

BOOL SuccessRead;

DWORD d = 0;//переменная, в которой будет храниться значение числа, передаваемого от клиента

while (TRUE)

{

hNamedPipe = CreateNamedPipeA( //создание канала

lpszPipeName, //имя канала

PIPE\_ACCESS\_DUPLEX, //режим доступа к каналу (односторонний/двусторонний)

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, //режимы работы канала: передавать сообщения|читать сообщения|ждать

PIPE\_UNLIMITED\_INSTANCES, //количество водящих соединений к каналу. в данном случае неограничено

SIZE\_BUFFER, // объем буфера на чтение (байт)

SIZE\_BUFFER, // объем буфера на запись (байт)

INFINITE, // максимальное время ожидания сообщения

NULL); //указатель на структуру безопасности

Connected = ConnectNamedPipe(hNamedPipe, NULL); //установка соединения клиента с каналом

if (Connected) //если клиент подключился

{

// printf("\nКлиент успешно подключился \n");

SuccessRead = ReadFile(hNamedPipe, buffer, size\_buffer, &actual\_readen, NULL);

if (SuccessRead)

{

printf("\nКлиент пишет: ");

printf(buffer);

printf("\n");

//отвечаем клиенту

printf("\nвведите сообщение для клиента:\n");

gets(message);

buffer = &message;//строковая переменная, значение которой записывается в канал

WriteFile(hNamedPipe, buffer, size\_buffer, &actual\_readen, NULL);

}

}

else

{

printf("\nКлиент отключился от сервера\n");

}

CloseHandle(hNamedPipe);//закрываем канал

}

}

**Клиент**

#include <stdio.h>

#include <Windows.h>

#define SIZE\_BUFFER 512

int main()

{

system("chcp 1251");

LPSTR lpszPipeName = "\\\\.\\pipe\\MyPipe";//имя канала (такое же, как и на сервере)

BOOL flag\_otvet = TRUE;

char message[SIZE\_BUFFER];

DWORD size\_buffer = SIZE\_BUFFER;//размер буфера для записи

DWORD actual\_written; //сколько на самом деле было записано байт

LPSTR buffer;

DWORD actual\_readen;

BOOL SuccessRead;

while (TRUE)

{

char message[SIZE\_BUFFER];

HANDLE hNamedPipe = CreateFileA(//открываем канал. по аналогии с открытием файла

lpszPipeName, GENERIC\_READ | GENERIC\_WRITE,

0, NULL, OPEN\_EXISTING, 0, NULL);

DWORD dwMode = PIPE\_READMODE\_MESSAGE;

BOOL isSuccess = SetNamedPipeHandleState(hNamedPipe, &dwMode, NULL, NULL);

if (!isSuccess)

{

printf("сервер не отвечает\n");

flag\_otvet = TRUE;

}

else

{

// printf("соединение установлено\n");

if (flag\_otvet)

{

printf("введите сообщение для сервера:\n");

gets(message);

buffer = &message;//строковая переменная, значение которой записывается в канал

WriteFile(hNamedPipe, buffer, size\_buffer, &actual\_written, NULL);

flag\_otvet = FALSE;

}

buffer = (CHAR\*)calloc(size\_buffer, sizeof(CHAR));

SuccessRead = ReadFile(hNamedPipe, buffer, SIZE\_BUFFER, &actual\_readen, NULL);

if (SuccessRead)

{

printf("\nСервер пишет: ");

printf(buffer);

printf("\n");

flag\_otvet = TRUE;

}

}

Sleep(100);

CloseHandle(hNamedPipe);//закрываем подключение к каналу

}

}

## Задание 23

1. **Создание линейного односвязного списка из n элементов. Значение элемента генерируется случайным образом. Поместить в отдельные указатели адрес минимального и максимального элемента списка. Вывести список и значения минимального и максимального элемента через указатель**

#include <Windows.h>

#include <stdio.h>

#include <locale.h>

typedef struct LOS {

int arg;

int min;

int max;

struct LOS\* next;

}LOS;

void GetMinMax(LOS\* los) {

LOS\* head = los;

int max;

int min = los->arg;

while (los) {

if (los->next != NULL) {

int n = los->next->arg;

if (los->arg > n)

max = los->arg;

}

if (min > los->arg)

min = los->arg;

los = los->next;

}

los = head;

los->min = min;

los->max = max;

}

void WriteLOS(LOS\* los) {

LOS\* head = los;

while (los)

{

printf("значение - %d\n", los->arg);

los = los->next;

}

los = head;

printf("min - %d; max - %d", los->min, los->max);

}

LOS\* createLOS(int count) {

LOS\* los = calloc(1, sizeof(LOS));

los->arg = rand();

los->next = NULL;

LOS\* headCopy, \* next;

headCopy = los;

for (size\_t i = 0; i < count - 1; i++)

{

next = calloc(1, sizeof(LOS));

next->arg = rand();

headCopy->next = next;

headCopy = next;

}

headCopy->next = NULL;

return los;

}

main() {

setlocale(LC\_ALL, "Rus");

LOS\* los = createLOS(5);

GetMinMax(los);

WriteLOS(los);

}

## Задание 24

1. **Программа для считывания числового значения из файла (с сохранением его в числовом формате). Исходный файл является бинарным**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

int main()

{

FILE\* file = NULL;

int number;

file = fopen("bin.bin", "wb");

if (file == NULL) {

printf("Error opening file");

}

scanf("%d", &number);

fwrite(&number, sizeof(int), 1, file);

fclose(file);

file = fopen("bin.bin", "rb");

if (file == NULL) {

printf("Error opening file");

}

fread(&number, sizeof(int), 1, file);

printf("%d", number);

fclose(file);

}

## Задание 25

1. **Создание консольной утилиты, которая печатает список переданных ей аргументов из командной строки. При демонстрации работы программы рассмотреть не менее трех способов передачи аргументов в запускаемый процесс**

## //Задано Задание 26

1. **Создание функции, аргументом которой является указатель на функцию. Создать не менее двух вспомогательных функций и передать их в качестве аргумента исходной функции. Продемонстрировать специфику работы исходной функции в зависимости от переданных параметров**

#include <stdio.h>

int kv(int a)

{

return a\*a;

}

int kub(int a)

{

return a\*a\*a;

}

int chto\_to(int (\*f)(int), int n)

{

return f(n);

}

int main(void)

{

system("chcp 1251>nul");

printf("1 - квадрат, 2 - куб ");

int (\*f)(int);

int i=0, n=0;

scanf("%d %d", &i, &n);

switch (i)

{

case 1:

f = kv;

break;

case 2:

f = kub;

break;

default:

f = kub;

break;

}

printf("\n%d", chto\_to(f,n));

return 0;

}

////////////////////////////////////////////////////////////////////

#include <stdio.h>

void kv(int\* a)

{

\*a \*= \*a;

}

void kub(int\* a)

{

\*a \*= \*a \* \*a;

}

int main(void)

{

system("chcp 1251>nul");

int a = 2;

kub(&a);

printf("%d", a);

return 0;

}

## Задание 27

1. **Использование анонимного канала для передачи строковых значений между процессами**

**Основной код программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

int main()

{

HANDLE hReadPipe, hWritePipe;

char message[] = "Hello, child process!";

char buffer[256];

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

DWORD bytesWritten, bytesRead;

BOOL success;

// Создаем анонимный канал

success = CreatePipe(&hReadPipe, &hWritePipe, &sa, 256);

if (!success)

{

printf("Error creating pipe\n");

return 1;

}

// Создаем дочерний процесс

STARTUPINFO si;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

si.hStdError = hWritePipe;

si.hStdOutput = hWritePipe;

si.dwFlags |= STARTF\_USESTDHANDLES;

PROCESS\_INFORMATION pi;

ZeroMemory(&pi, sizeof(pi));

success = CreateProcessA(NULL, "..\\Debug\\Child.exe", NULL, NULL, TRUE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi);

if (!success)

{

printf("Error creating child process\n");

return 1;

}

while (TRUE) {

// Читаем из канала

success = ReadFile(hReadPipe, buffer, sizeof(buffer), &bytesRead, NULL);

if (!success)

{

printf("Error reading from pipe\n");

return 1;

}

buffer[bytesRead] = '\0';

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

}

}

**Второй проект: создаем с названием Child, если другое, меняем путь**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

int main()

{

HANDLE hReadPipe = GetStdHandle(STD\_OUTPUT\_HANDLE);

while (TRUE)

{

char message[256];

DWORD bytesWritten;

scanf("%s", message);

WriteFile(hReadPipe, message, strlen(message), &bytesWritten, NULL);

}

return 0;

}

# Теория

## еЗадание 1

Задание 1

1. Windows HOOK. Назначение и особенности использования.

Хуками называются перехватчики системных событий. Они позволяют программно подменять обработчик этих событий.

Назначение:

Хуки необходимы для считывания нажатий и корректной обработки их.

Хуки используются при любом нажатии и событии в системе Windows. Хуки используются, например в вижуал студии при создании события он клик.

Любое событие в системе может вызвать другое при обработке хуком

Особенности использования:

За установку хука отвечает функция SetWindowsHookEx Она возвращает Указатель на структуру HHOOK.

Для того чтобы интерпретировать события нажатия виртуальных клавиш используется функция TranslateMessage.

Для отправки сообщения в обработчик хука используется функция DispatchMessage.

Функция UnhookWindowsHookEx(hHook); освобождает дескриптор хука.

## Задание 2

2. Актуальность синхронизации потоков при параллельной обработке данных

Поток – последовательный набор инструкций, некий объект исполняемого процесса.

Актуальность синхронизации потоков при параллельной обработке данных заключается в том, что она исключает возможность неправильного поведении программы. Например, когда несколько потоков получают доступ к одному ресурсу, значение может быть записано или прочитано не вовремя.

Когда потоки пытаются использовать один ресурс одновременно, программа может работать не корректно. Например, при одновременно увеличении переменной на 1 разными потоками, некоторые могут не выполнить необходимые действия, так как его выполняет другой поток. Значение может быть записано или прочитано раньше времени.

Существует много способов синхронизации и потоков, главным принципом является запрет параллельного доступа к общим ресурсам. Например, при помощи создания критической секции.

## //Задано Задание 3

3. Библиотеки DLL. Назначение и использование DLL-библиотек

Динамическая библиотека содержит код функций, который может быть использован в нескольких программах одновременно. Использование DLL-библиотек позволяет экономить оперативную память, так она подключается к проекту только в момент ее вызова, и отключается после того как пропала необходимость в ней. Для работы с динамической библиотекой, ее необходимо подключить с помощью функции, далее импортировать функции и в окончании отключить библиотеку.

Большие проекты имеют огромное количество функций, которые могут использоваться в нескольких местах. Но не все функции используются при выполнении, но занимают место в ОП и увеличивают размер исполняемого файла.

Решения могут содержать много проектов, копировать код не лучший выбор, так как он не универсален. Эти проблемы решают библиотеки (статические и динамические).

DLL (динамические) подключаются к проекту и отключаются в нужный момент времени. Таким образом экономится ОП. Например, можно не описывать выстрел из пистолета 100 раз в проекте, можно брать из DLL меняя параметры под конкретное оружие

## Задание 4

4 Многопоточный режим работы компьютера. преимущества использования многопоточности

В основе парадигмы программирования лежит понятие процесс.

Процесс можно понимать как некий объект для исполняемого файла. один и тот же исполняемый файл может быть запущен несколько раз (при этом создаётся несколько процессов).

Вся информация о процессе хранится в оперативной памяти.

Процесс характеризуется неким набором информации. За работу процессор отвечают потоки.

Поток рассматривается как последовательный набор инструкций, Который выполняется на процессоре. Каждый процесс имеет хотя бы один поток. Данный поток называется основным

Процессы могут находиться в одном из следующих состояний:

• Выполняется. Основному потоку выделено процессорное время. Количество выполняемых процессов одновременно и не может быть больше, чем Потоков, который поддерживает процессор

• Состояние готовности к выполнению. Процессу Предоставлены все ресурсы, кроме процессорного времени.

• Состояние ожидания. Процессу предоставлены не все ресурсы (Например идёт ввод или вывод данных).

За время своего существования один и тот же процесс может многократно менять свои состояния.

Место процесса в очереди определяется его приоритетом.

Приоритет процесса может поменять приоритет (в том числе и создаться) в одном из следующих случаев:

• По команде пользователя.

• При выборе из очереди планировщиком операционной системы

• По таймеру системному

• По инициативе другого процесса

При создании процесса создаётся его дескриптор, который является структурой, содержащий всю необходимую информацию о процессе.

Основной поток в процессе может также порождать и вспомогательные потоки.

Они нужны для параллельного выполнения операций.

Плюсы Минусы

Переключение между выполняющимися процессами потребляет заметную долю временных и других ресурсов ОС, а в случаях, аналогичных многопроцессному поиску (grepMP, программа 6.1), все процессы заняты выполнением одной и той же программы. Организация параллельной обработки файла с помощью потоков в контексте единственного процесса позволяет снизить общие накладные расходы системы. Поскольку потоки разделяют общую память и другие ресурсы, принадлежащие одному процессу, существует вероятность того, что один поток может случайно изменить данные, относящиеся к другому потоку.

С использованием только однопоточных процессов трудно организовать простое и эффективное управление несколькими параллельно выполняющимися задачами, взаимодействующими между собой, в таких, например, случаях, как ожидание и обработка пользовательского ввода, ожидание ввода из файла или сети и выполнение вычислений. При определенных обстоятельствах вместо улучшения производительности может наблюдаться ее резкое ухудшение.

Улучшенная реакция приложения - любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой.

Создание многопоточной программы требует очень тщательной разработки. Вероятность появления незначительных временных сбоев или ошибок, вызванных нечаянным совместным использованием переменных, в такой программе весьма значительна. Алан Кокс (Alan Сох, всеми уважаемый гуру Linux) сказал, что потоки равнозначны умению "выстрелить в обе собственные ноги одновременно".

Улучшенная структура программы - некоторые программы более эффективно представляются в виде нескольких независимых или полуавтономных единиц, чем в виде единой монолитной программы. Многопоточные программы легче адаптировать к изменениям требований пользователя. Отладка многопоточной программы гораздо труднее, чем отладка одного потока исполнения, поскольку взаимосвязи потоков очень трудно контролировать.

переключение между потоками требует от операционной системы гораздо меньше усилий, чем переключение между процессами. Программа, в которой громоздкие вычисления разделены на две части, и эти две части выполняются как отдельные потоки, необязательно будет работать быстрее на машине с одним процессором, если только вычисление не позволяет выполнять обе ее части одновременно и у машины, на которой выполняется программа, нет многоядерного процессора для поддержки истинной многопоточности.

## Задание 5

5. Особенности использования строкового типа данных. Массивы строк.

Строка - это последовательность ANSI или UNICODE символов.

Строковый тип данных в C как таковой отсутствует, а в качестве строк в С используются обычные массивы символов.

Исторически сложилось два представления формата строк:

1. формат ANSI;

2. cтроки с завершающим нулем (используется в C).

Формат ANSI устанавливает, что значением первой позиции в строке является ее длина, а затем следуют сами символы строки.

В строках с завершающим нулем, значащие символы строки указываются с первой позиции, а признаком завершения строки является значение ноль.

Так как строки на языке С являются массивами символов, то к любому символу строки можно обратиться по его индексу. Для этого используется синтаксис обращения к элементу массива, поэтому первый символ в строке имеет индекс ноль.

Объявление массивов строк в языке С также возможно. Для этого используются двумерные массивы символов, что имеет следующий синтаксис:

Char имя[количество][длина];

Первым размером матрицы указывается количество строк в массиве, а вторым – максимальная (с учетом завершающего нуля) длина каждой строки. Например, объявление массива из пяти строк максимальной длиной 30 значащих символов будет иметь вид:

char strs[5][31];

Для ввода и вывода строковой информации можно использовать функции форматированного ввода и вывода (printf и scanf). Для этого в строке формата при вводе или выводе строковой переменной необходимо указать спецификатор типа %s.

В С для преобразования строк, содержащих числа, в численные значения в библиотеке stdlib.h предусмотрен следующий набор функций:

double atof(const char \*string); // преобразование строки в число типа double

int atoi(const char \*string); // преобразование строки в число типа int

long int atol(const char \*string); // преобразование строки в число типа long int

long long int atoll(const char \*string); // преобразование строки в число типа long long int.

Вообще говоря, символ (символ) занимает 1 байт или 2 байта, мы говорим, что 1-байтовый символ является ANSI, он может использоваться для представления английских букв, а 2 байта, которые мы называем Unicode, могут представлять Все языки в мире – префикс L.

VC ++ использует встроенные типы данных char и wchar\_t для представления символов ANSI и Unicode соответственно.

char\*Замена:LPSTR - это (Длинный) указатель на строку.

const char\*Замена:LPCSTR

WCHAR\*Замена:LPWSTR

const WCHAR\*Замена:LPCWSTR

TCHAR\*Замена:LPTSTR

const TCHAR\*Замена:LPCTSTR

'L' означает wchar\_t, который, в отличие от нормального символа, требует 16 бит хранения, а не 8 бит. Вот пример:

"A" = 41

"ABC" = 41 42 43

L"A" = 00 41

L"ABC" = 00 41 00 42 00 43

A wchar\_t в два раза больше, чем просто char. В ежедневном использовании вам не нужно использовать wchar\_t, но если вы используете windows.h, вам это понадобится..

## Задание 6

6. Особенности настройки проекта и использования синтаксиса языка программирования при написании DLL-файла

Для создания библиотеки DLL нужно добавить соответствующий проект в решение и настроить его на запуск как динамическую библиотеку.

Проект DLL имеет некоторые особенности.

Он настраивается также, как и проект winapi. Ну, помимо этого ещё необходимо указать в настройках проекта на вкладке общие Тип конфигурации динамическая библиотека.

Точка входа у библиотеки DLL тоже своя особенная:

BOOL WINAPI DllMain(мне кажется это достаточно)

(HINSTANCE hlnstDll, DWORD dwReason, LPVOID IpReserved)

{

BOOL bAllWentWell = TRUE;

switch (dwReason)

{

case DLL\_PROCESS\_ATTACH:

break;

case DLL\_THREAD\_ATTACH:

break;

case DLL\_THREAD\_DETACH:

break;

case DLL\_PROCESS\_DETACH:

break;

}

if (bAllWentWell)

return TRUE;

else

return FALSE;

}

Все экспортируемые из DLL функции должны иметь специальные соглашения о вызовах \_cdecl. (Дело в том, что приложение winapi по умолчанию имеет соглашения вызовах \_\_stdcall).

Также же функции необходимо пометить с помощью специального оператора:

\_\_declspec(dllimport) - для Импортируемых функций

\_\_declspec(dllexport) - для экспортируемых функций.

И это всё описывается в прототипе функции. Без прототипа мы её импортировать или экспортировать не сможем (Но просто писать функции в dll мы можем).

## Задание 7

7. Особенности разработки программ в WinAPI

Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системы MS Windows. Работа через Windows API — это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ.

Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си.

В вин апи отличается соглашение о вызовах.

Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в программах, исполняемых в среде Windows, могут пользоваться этим API. Пример: C++, C#, Pascal, Visual Basic.

Функция запуска приложений Windows

Функция называется WinMain. Принимает 4 параметра. они описаны ниже

int WINAPI WinMain(HINSTANCE hlnstance. // дескриптор, присваиваемый запущенному приложению

HINSTANCE hPrevInstance, // для совместимости с winl6. в Win32 не используется

LPSTR lpCmdLine. // указатель на командною строку, если приложение так запущено

int nCmdShow); // значение, которое может быть передано в функцию Show Window ()

Перед запуском приложений Windows необходимо особым образом настроить проект

Также при работе с API системы Windows необходимо подключать заголовочный файл windows.h

С помощью вин апи мы можем работать с системными окнами виндовс

## Задание 8

8. Особенности разработки программ для работы с системными событиями использования мыши и клавиатуры. (Та же хуета, как в первом вопросе)

## Задание 9

9. Подходы к обработке исключений в программах WinAPI, написанных на си

• использование возвращаемых значений функций и последующая их проверка, анализ и т.д.;

• использование исключений и обработка исключительных ситуаций;

Исключение возникает только в случае непредвиденной ошибки времени выполнения и обрабатывает все остальное с кодами ошибок и результатов — напрямую и близко к источнику сбоя. Таким образом, при вызове исключения вы будете знать, что его причина заключается в ошибке в вашем коде или в состоянии исключения системы.

Обработка ошибок на основе анализа возвращаемого значения производится вне метода, в котором возникла ошибка. Т.е. для использования такого подхода вы должны быть уверены, что метод хоть что-то вернёт, какая бы ошибка внутри него не возникла. Велика вероятность, что вы об этом не узнаете: если возникнет ошибка, ваша программа может быть просто остановлена системой.

Механизм обработки ошибок на основе исключений работает внутри метода и, если обработчик исключения в этом методе не найден, то распространяется на следующий метод в стеке (тот, из которого был вызван метод, в котором произошло исключение) и т.д. И только если обработчик исключения так и не будет найден, система аварийно завершит выполнение вашей программы.

## Задание 10

10. Понятие “системное программирование”. предназначение и специфика системного программирования

Системой называется отношение двух множеств: множество элементов и множество связей между этими элементами.

Программирование — это процесс написания программного кода.

Системные программисты пишут программные коды для функционирования системы

Операционная система выступает посредником между пользователем и аппаратной частью компьютера.

Основная отличительная черта системного программирования по сравнению с прикладным программированием заключается в том, что результатом прикладного является выпуск программного обеспечения, предлагающего определенные услуги пользователям (текстовый процессор). Результатом системного программирования является выпуск программного обеспечения, предлагающего сервисы по взаимодействию с аппаратным обеспечением (дефрагментация жёсткого диска), что подразумевает сильную зависимость таких программ от аппаратной части.

## Задание 11

11. Предназначение директив препроцессора и заголовочных файлов

Серьёзный программные продукты имеют достаточно большое количество строчек кода. Если весь программный код помещается в одном файле, то его восприятие становится затруднительным.

Поэтому логически завершенные фрагменты программного кода рекомендуется помещать в отдельные файлы.

Так как множество строк кода приходится повторять в разных файлах, такие как объявление глобальных переменных и указание прототипов функций, их обычно выносят в заголовочные файлы

Такие файлы называются файлами заголовков.

Файлы заголовков имеет расширение .h и добавляется в проекте стандартным способом (с помощью #include).

Пользовательские файлы заголовков указывается в двойных кавычках.

Файлы заголовков рекомендуется использовать для объявления объектов и не рекомендуется для их инициализации.

Директивы препроцессора представляют собой инструкции, записанные в тексте программы на СИ, и выполняемые до трансляции программы. Директивы препроцессора позволяют изменить текст программы, например, заменить некоторые лексемы в тексте, вставить текст из другого файла, запретить трансляцию части текста и т.п.

Препроцессор - это специальная программа, которая осуществляет алгоритмические действия перед компиляцией основного кода.

Команды для препроцессора называются директивами.

Директивы начинаются с символа #, В конце строки директивой “;” можно не ставить.

Список основных директив:

#include. Вставляет содержимое из текстового файла в то место, где она написана

#define. Имеет три основных применения:

• Инициализация параметров (Задание флагов). Используется преимущественно для условной компиляции

• Задание констант. Используется в качестве альтернативы глобальным переменным

• Задание макроопределений (макросов)

#undef. Отменяют задание параметра

#ifdef - Условие компиляции, если определённый параметр задан

#ifndef - Условие компиляции, если определённый параметр не задан

#if - Инициализация условной компиляции. далее необходимо ввести условие, используя другие Директивы препроцессора.

#elif - Директивы для создания вложенных условий.

#else - Ветка при ложности всех условий. Её нельзя ставить выше чем #elif

#error - Внесение искусственной ошибки для компиляции

## Задание 12

12. Принципы взаимодействия операционной системы с прикладными программами (в том числе раскрыть понятия “процесс”, “поток”, “дескриптор”)

Операционная система выступает посредником между пользователем и аппаратной частью компьютера. Где каждый процесс — это отдельно взятая запущенная программа.

Windows выступает «посредником» между программой и устройствами компьютера.

Процесс можно понимать как некий объект для исполняемого файла. один и тот же исполняемый файл может быть запущен несколько раз (при этом создаётся несколько процессов).

Вся информация о процессе хранится в оперативной памяти.

Процесс характеризуется неким набором информации. За работу процессор отвечают потоки.

Поток рассматривается как последовательный набор инструкций, Который выполняется на процессоре. Каждый процесс имеет хотя бы один поток. Данный поток называется основным.

Место процесса в очереди определяется его приоритетом.

Дескриптор (HANDLE) — это идентификатор объекта, который создается не вашей программой, а операционной системой или сторонней библиотекой.

При создании процесса создаётся также его дескриптор, который является структурой, содержащий всю необходимую информацию о процессе.

## //Задано Задание 13

13. Принципы функционирования системного программного обеспечения

Системное программное обеспечение — это комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование, выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя.

В отличие от программ, которые взаимодействуют непосредственно с пользователем (прикладное программное обеспечение), системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и микропрограммной реализации вычислительной системы, управляет аппаратными ресурсами вычислительной системы.

Оно разрабатывается так, чтобы компьютер мог эффективно выполнять прикладные программы.

Также к таким программам относятся драйвера, которые настраивают определённую аппаратную часть компьютера (Например: драйвера на видеокарту, мышку и тд)

Главной особенностью функционирования системного программного обеспечения как драйвер можно назвать то, что у него отсутствует графический интерфейс взаимодействия с пользователем, поскольку данная программа предназначена только для того, чтобы сообщить компьютеру как нужно обращаться с этим устройством и не требует никакого взаимодействия с пользователем после установки для корректной работы.

Не менее важной особенностью системного программного обеспечения является то что данное программное обеспечение пишется на языка низкого уровня. Из этого следует то что системное программное обеспечение управляет ресурсами компьютером напрямую и имеет наибольшие возможности управления компьютером и его составляющими (ОЗУ, сетевая карта), по сравнению с прикладными программами.

Также из этого следует то что что системное программное обеспечение работает в разы быстрее поскольку отсутствуют некоторые уровни абстракции, которые присутствуют в языках высокого уровня, которые предназначены для упрощения написания программ для решения прикладных задач.

## Задание 14

14. Реестр Windows. использование системного реестра программами

Формально реестр представляет из себя базу данных.

Элементами этой базы данных являются параметры. Организация параметров представляет из себя древовидную иерархию.

Чаще всего реестр используется для хранения параметров операционной системы или прикладных программ.

Реестр задумывался как некое общее хранилище настроек. До этого программы хранили свои настройки в конфигурационных файлах. Использование конфигурационных файлов имела ряд недостатков:

• Не было централизованного хранилища этих файлов.

• Проблема защиты файлов:

− Текстовые файлы можно было удалить

− Можно было изменить содержимое текстовых файлов по-своему

Для работы с реестром в системе Windows предусмотрена специальная утилита RegEdit.EXE

Сам реестр имеет 5 глобальных веток:

Вообще механизм его работы достаточно прост. При установке программы (или изменении какого-либо параметра Windows в любом из меню настроек) система сама разыскивает нужные параметры и вносит коррективы в одно из значений реестра. Программно можно изменить некоторые параметры, например шрифт, используемый в приложении.

Хранение паролей для прикладных программ (практически нет никаких шансов получить какой-либо пароль с помощью простого просмотра данных реестра. Пароли, в подавляющем большинстве случаев, хранятся в зашифрованном виде, и для их дешифрации потребуется специальное программное обеспечение).

## Задание 15

15. Системный буфер обмена. Особенности взаимодействия программного обеспечения с системным буфером обмена

По сути буфер обмена можно рассматривать как канал или глобальный указатель, к которому имеют доступ все прикладные программы.

Однако при работе с буфером следует учитывать и его недостатки:

• Буфер обмена только один в системе. и любое приложение может его перезаписать. Поэтому нет гарантии целостности получения данных

• Информация не может быть приватной (Во сколько буфер обмена доступен всем)

Буфер обмена на позволяет хранить данные различных типов. Однако при работе с ним программы учитывают этот контекст.

В зависимости от формата данных в буфере с ним могут взаимодействовать те или иные приложения.

Работа с буфером обмена

Буфер обмена нужно скорее воспринимать как указатель, а не как переменную.

У буфера нет своей заранее выделенной области. Поэтому память выделяется в самой программе.

Буфер обмена предоставляет набор возможностей для временного (не дискового) хранения информации. Данные, записанные в буфер обмена, могут передаваться между различными приложениями или возвращаться в исходное приложение, из которого они попали в буфер.

Приложение-источник может копировать данные в буфер обмена в одном из предопределенных или пользовательских форматов. Буфер самостоятельно управляет выделением памяти и размещением переданных ему данных. После того как данные попадут в буфер обмена, любое приложение сможет получить к ним доступ, определить тип данных и при желании скопировать их.

Когда программа просмотра Clipbrd.EXE является активной, она регулярно запрашивает буфер обмена о том, содержатся ли в нем данные и каков их тип. Затем, если это возможно, программа копирует данные из буфера и отображает их в своем окне.

Так как буфер обмена только один, все приложения должны использовать его совместно. Но совместное использование неизбежно связано с возможностью конфликтов. Предположим, что приложение А записало в буфер обмена растровое изображение, а затем приложение Б записало в него блок текстовых данных. Поскольку приложение Б вполне закономерно начинает с того, что очищает буфер обмена, картинка, записанная приложением А. удаляется. Если приложение В. Для которого была предназначена эта картинка, уже успело ее прочитать, то все в порядке. Если же приложение В не скопировало изображение до того, как приложение Б записало в буфер обмена текст, картинка пропадает.

Общедоступность буфера обмена также может стать причиной возможных ошибок. Поскольку элемент данных, записанный в буфер, не может быть адресован какому-либо конкретному приложению. доступ к этой информации может быть ошибочно получен другим приложением, которое работает с данными того же типа.

Буфер обмена может содержать данные нескольких типов, записанных одним или несколькими приложениями. В этом случае проблема заключается в том, как различить эти блоки (например, несколько блоков текста, записанных из разных источников). По этой причине приложение обычно очищает буфер обмена перед тем, как записывать в него новый материал.

Эти факторы необходимо учитывать, но они не представляют серьезной проблемы. Если перечисленные недостатки накладывают дополнительные ограничения на работу программы, можно воспользоваться технологиями каналов (pipe).

Управление глобальными блоками памяти, которые содержат данные, помещенные в буфер обмена, осуществляется с помощью флагов выделения памяти. При записи информации в буфер программа выделяет блок памяти с помощью функции GlobalAlloc() и флага GHND (определенного как GMEM MOVABLE | GMEM ZEROINIT).

Обычно при закрытии исходного приложения выделенная ему глобальная область памяти удаляется (освобождается) операционной системой. Но сели приложение вызовет функцию SetClipboardDataO с указанием дескриптора глобального блока памяти. Windows возьмет этот блок в свою собственность, точнее, в собственность буфера обмена, изменив флаги выделения.

Принадлежность глобального блока памяти назначается функцией GlobalRealloc(): GlobalRealloc(hMem. NULL. G M E M M O D IF Y | G M E M D D E S H A R E ) ;

После этого вызова выделенный блок памяти не принадлежит исходному приложению и доступен только через буфер обмена с помощью функции GetClipboardData(). Эта функция предоставляет вызвавшему ее приложению временный доступ к данным, записанным в буфере обмена, передавая программе дескриптор глобального блока памяти. Однако принадлежность блока данных по-прежнему сохраняется за буфером обмена, а не передается приложению. Следовательно, данные, записанные в буфере обмена, могут быть удалены только путем вызова функции EmptyClipboard.

Хотя многие ресурсы Window's предназначены для совместного использования несколькими приложениями, доступ к буферу обмена в каждый момент возможен только одной программой. Это позволяет предотвратить конфликты между приложениями.

Прежде чем приложение начнет читать, записывать или удалять содержимое буфера обмена, необходимо запросить доступ к нему с помощью функции OpenClipboard(). Эта функция возвращает значение TRUE, если буфер открыт и доступ к нему разрешен, и значение FALSE, если доступ к буферу запрещен по той причине, что в данный момент право доступа принадлежит другому приложению.

Закончив работу с буфером обмена, приложение должно вызвать функцию CloseClipboard. которая делает буфер доступным для других программ. Следует отметить, что каждый вызов функции OpenClipboardO всегда должен сопровождаться вызовом функции CloseClipboard().

Функция TransferToClipboard() начинается с запроса на право доступа к буферу обмена. Затем в буфер копируется одиночный блок памяти. Наконец, функция закрывает буфер обмена, освобождая его для доступа другим приложениям.

Прежде чем начинать чтение информации из буфера обмена, необходимо определить, какого типа данные содержатся в нем. Поскольку для различных типов данных необходимо применять разные операции, приложение должно заранее знать, что именно ему придется читать, и подготовиться к определенному виду обработки. В частности, вы можете запросить данные определенного типа и посмотреть на результат. Однако такой подход не слишком элегантен, да и эффективность его не очень высока.

Более эффективный способ анализа содержимого буфера обмена заключается в использованииАРІ-функции IsClipboardFonnatAvailable() или EnumClipboardFormats().

Функция IsClipboardFonnatAvailableO возвращает булево значение, которое сообщает, содержит ли буфер обмена данные нужного формата.

Функция EnumClipboardFormats проверяет наличие данных всех возможных форматов. При первом вызове функции с параметром NULL oнa возвращает информацию о первом доступном формате. При каждом последующем вызове возвращаются сведения о других форматах. Таким образом, для получения списка форматов можно воспользоваться следующим алгоритмом:

wFormat = NULL;

OpenCli pboa rd( hwnd);

while(wFormat = EnumClipboardFormals(wFomiat))

{

...код обработки различных форматов...

}

CloscClipboardO;

Следует отметить, что на операции, которые могут выполняться с буфером обмена, налагается несколько ограничений:

1. Прежде чем копировать данные в буфер обмена, необходимо вызвать функцию EmptyClipboard(). предназначенную для удаления текущего содержимого. Сам факт доступа к буферу обмена еще не означает, что будет получен контроль над его содержимым. Фу нкция Empty Clipboard() позволяет стать владельцем буфера и одновременно удалить его текущее содержимое.

2. Любое приложение может получить доступ к содержимому буфера обмена, но лишь владелец буфера, т. е. приложение, вызвавшее функцию Empty Clipboard(). имеет право записывать в него данные. Поскольку владельцем буфера обмена может быть только одно приложение, данные, записанные предыдущим владельцем, удаляются, даже если таковым было это же самое приложение.

3. Хотя в буфер обмена может быть записано несколько блоков данных, передавать их нужно в течение одной операции. Буфер обмена нельзя открыть, записать в него данные, закрыть, а затем снова открыть и добавить другие данные, не удалив предыдущий фрагмент.

4. В буфере обмена одновременно может находиться только по одному элементу данных каждого типа. Это объясняется простой причиной: не существует способа разделения нескольких элементов данных одного и того же типа. Однако при наличии данных нескольких типов приложение, получившее доступ к буферу обмена, может запросить только один элемент, несколько элементов или все элементы. В этом случае данные каждого типа должны запрашиваться отдельно.

Буфер обмена может многократно открываться для запроса других элементов данных или повторного запроса того же элемента. Но в целом, запрашивая фрагмент данных из буфера обмена. лучше всего создать его локальную копию, а не повторять один п тот же запрос снова и снова.

Кроме того, не забывайте: нет никакой гарантии того, что при следующем запросе тот же самый элемент данных останется неизменным, поскольку буфер обмена является общедоступным.

## Задание 16

16. Структура системного реестра. Типы данных в системном реестре.

HKEY CLASSES ROOT— хранится информация о зарегистрированных классах, расширениях документов;

HKEY CURRENT USER — хранится информация о текущей пользовательской конфигурации, внешнем виде рабочего стола, сетевых настройках;

HKEY LOCAL MACHINE — хранится информация о системной и аппаратной конфигурации;

HKEY USERS — хранится информация обо всех зарегистрированных пользователях;

HKEY\_CURRENT\_CONFIG — текущая аппаратная конфигурация.

Крупные ветки называются ульями.

Более мелкие ветки называются ключами реестра.

Каждый ключ может содержать внутри себя другие ключи или параметры.

Параметры также имеют свои определенные типы:

• Строковый параметр. Содержит последовательность символов в определенной кодировке. Его идентификатор в системе REG\_SZ

• Двоичный параметр. По сути содержит массив байт. Его идентификатор в системе REG\_BINARY

• Машинное слово для 32-битных систем. Его идентификатор в системе REG\_DWORD

• Машинное слово для 64 разрядной системы. Его идентификатор в системе REG\_QWORD

• Мультистроковый параметр, другими словами, массив строк. Его идентификатор в системе REG\_MULTI\_SZ

• Расширяемый строковый параметр. По сути, представляет из себя строку переменной длины. REG\_EXPAND\_SZ

## Задание 17

17. Структуры данных. актуальность использования структур. Передача структуры как параметра.

Структура — это, некое объединение различных переменных (даже с разными типами данных), которому можно присвоить имя. Например, можно объединить данные об объекте Дом: город, улица, количество квартир и т.д. в одной структуре. В общем, можно собрать в одну совокупность обо всем, что необходимо конкретному программисту.

Структура также статический располагается в памяти, поэтому на неё тоже можно создать указатель.

Обращение к полю структуры через указатель осуществляется с помощью оператора Стрелка ->

Как и любой другой объект, структура может использоваться в качестве параметра функции и также может быть возвращаемым объектом функции.

В начале программы определена структура time, которая представляет время и содержит три элемента для хранения значений для часов, минут и секунд.

Указатели на структуру как параметры

При использовании структур в качестве параметров в функции следует учитывать, что при вызове функции для структуры, также как и для параметров типа int или char, выделяется память, в которую помещаются значения элементов структуры. То есть структура в функцию передается по значению, а это значит, что переданную в функцию структуру мы изменить не можем.

Если мы действительно хотим изменить структуру в функции, то на надо передавать не саму структуру, а указатель на нее:

В итоге любое изменение значений по этому указателю приведет к изменению самой структуры. И результаты программы будут уже отличаться.

## Задание 18

18. Типы данных в ОС Windows

Беззнаковые целые числа представляются в виде последовательности битов в диапазоне от 0 до 2n-1, где n-количество занимаемых битов.

Знаковые целые числа представляются в диапазоне -2n-1…+2n-1-1. При этом старший бит данного отводится под знак числа (0 соответствует положительному числу, 1 – отрицательному).

• Тип BYTE обозначает 8-разрядное беззнаковое символьное значение. Тип byte — это знаковый 8-битовый тип. Его диапазон — от -128 до 127. Он лучше всего подходит для хранения произвольного потока байтов, загружаемого из сети или из файла.

• Тип WORD — 16-разрядное беззнаковое короткое целое.

• Тип DWORD — беззнаковое длинное целое.

• Тип UINT — беззнаковое 32-разрядное целое.

• Тип LONG эквивалентен типу long. Тип long предназначен для представления 64-битовых чисел со знаком. Его диапазон допустимых значений достаточно велик даже для таких задач, как подсчет числа атомов во вселенной.

• Тип BOOL обозначает целое и используется, когда значение может быть либо истинным, либо ложным.

• Тип LPSTR определяет указатель на строку.

• Тип LPCSTR определяет константный (const) указатель на строку.

• Тип HANDLE обозначает 32-разрядное целое, используемое в качестве дескриптора (Дескриптор - это указатель или индекс, к которому не привязан видимый тип). Это просто число, определяющее некоторый ресурс.

Дескриптор выступает в качестве идентификатора определённого ресурса

## //Задано Задание 19

19. Указатель как тип данных. особенности работы с указателями

Указатель представляет из себя переменную, значением которой является адрес другой переменной.

Под указатель тоже выделяется место в оперативной памяти. Количество байт зависит от разрядности программы.

Переменные однозначно идентифицируется двумя параметрами: Это её адрес (Номер 1 ячейки памяти)

К указателям применима операции сложения и вычитания. Они интерпретируются как сдвиги на определённое количество ячеек памяти. Шаг равен количество байтов, на которые ссылается указатель.

## Задание 20

20. Массив как тип данных. Особенности использования массивов при разработке программ на си

Фактически массив является указателем на выделенную область в памяти (количество элементов \* на размер элемента(тип данных)).

Имя массива является указателем на первый элемент. А объём памяти равен суммарному объему Всех элементов с учетом их типа.

В Си можно создавать динамические массивы. За динамическое выделение памяти отвечает функция malloc(), calloс(),realoс();

malloc(), В качестве аргумента данная функция принимает количество байт, которые нужно выделить.

Calloс()В качестве аргумента данная функция принимает количество элементов, и размер одного элемента(зануляет все выделенные ячейки памяти)

Realoс() изменяет ранее заданный размер области памяти с сохранением данных

За освобождение памяти отвечает функция free(). в качестве аргумента функция принимает указатель

Функция sizeof() принимает в качестве аргументов идентификатор типа, А возвращается количество байт, которое он занимает.

## Задание 21

21. Линейный односвязный список. Особенности создания и примеры использования

Линейный односвязный список – самая простая из динамических структур. Каждый элемент списка – структура, в которой есть поле с адресом на другие элементы списка, а другая часть полей – данные

Особенностью создания является то, что, так как это структура, то сектора в памяти выделяются не подряд. С использованием динамических структур можно проще вносить изменения в середину списка, так как при этом используются только соседние элементы.

Особенность создания заключается в том, что элемент не считается завершенным пока он не имеет связи со следующим элементом

Примеры использования разнообразны, так как-то какие данные хранить в структуре выбирает программист. Наиболее удобной структура будет если необходимо поменять элементы местами или удалить элемент из середины без необходимости сдвига остальных элементов

## //Задано Задание 22

22. Указатель на функцию. Назначение и примеры использования

В языке программирования C функция имеет адрес и может иметь указатель. Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции.

Самым распространенным указателем на функцию является ее имя. С помощью имени функции мы можем вызывать ее и получать результат ее работы.

Под функцию, также, как и под переменную выделяется память. соответственно можно создать указатель на ту область памяти, в которой находится эта функция.

По аналогии с массивом, имя функции является указателем на первую ячейку памяти, в которой она находится:

Указателю на функцию можно присвоить функцию, которая соответствует указателю по возвращаемому типу и спецификации параметров.

Если на уровне среды программирования или непосредственно в Си-программах имеет место элемент динамического связывания, то с уверенностью можно сказать, что в том или ином виде (явно или неявно) используется указатель на функцию:

• DLL, а также все возможные виды динамической загрузки внутреннего (исполняемого) программного кода;

Указатели широко используются в C в основных целях:

• передача функций в другие функции

• для итерации элементов в массивах или других структурах данных.

## Задание 23

23. Файл как тип данных. Использование файлов в работе программного обеспечения

В программе могут быть использоваться файлы для ввода и вывода. Они находятся в файле <stdio.h>. В работе ПО файлы можно открыть, закрыть, прочитать и записать. Переменная типа файл представляет собой указатель на файловый поток. За открытие файла отвечает функция foopen. Существует несколько режимов доступа к файлу, например, только для чтения, только для записи, для обновления. Файлы можно открыть в бинарном режиме, для этого необходимо при создании к режиму в конце добавить b. Для открытия в текстовом режиме, необходимо b заменить на t.

Файл можно читать и файл можно записать. Для взаимодействия с ним нужно создать переменную – указатель на файловый поток, главные параметры, имя файла и пусть к нему и режим доступа. Если правильно распарсить в файле можно хранить числовые, текстовые данные. Также данные можно записать в структуру с определенными полями. Важно следить за размером буфера и соблюдать кодировки.

Файловый ввод-вывод Находится также в заголовочном файле <stdio.h>

Типовыми функциями для работы с файлами являются следующие:

• Создание указателя на файл (Открытие файла)

• закрытие файла

• вывод информации из файла (чтение)

• Запись информации в файл

Для открытия файла в системе C необходимо объявить переменную типа файл (Точнее указатель на файловый поток)

За открытие файла отвечает функция fopen.

Данная функция принимает два параметра:

1. Имя файла (путь к нему)

2. Режим доступа к файлу (имеется в виду чтение или запись)

Список режимов:

r Чтение. Файл должен существовать.

w Запись нового файла. Если файл с таким именем уже существует, то его содержимое будет потеряно.

a Запись в конец файла. Операции позиционирования (fseek, fsetpos, frewind) игнорируются. Файл создаётся, если не существовал.

r+ Чтение и обновление. Можно как читать, так и писать. Файл должен существовать.

w+ Запись и обновление. Создаётся новый файл. Если файл с таким именем уже существует, то его содержимое будет потеряно. Можно как писать, так и читать.

a+ Запись в конец и обновление. Операции позиционирования работают только для чтения, для записи игнорируются. Если файл не существовал, то будет создан новый.

Если необходимо открыть файл в бинарном режиме, то в конец строки добавляется буква b, например “rb”, “wb”, “ab”, или, для смешанного режима “ab+”, “wb+”, “ab+”. Вместо b можно добавлять букву t, тогда файл будет открываться в текстовом режиме.

Функция для чтения данных из файла:

За это отвечает функция ReadFile. Аргументы у неё такие же как его функции WriteFile,

Но при чтении есть некоторые особенности:

Надо указывать заведомо большой буфер для чтения (Так как мы не знаем Сколько информации находится в файле)

После прочтения строку нужно закрыть (Дописать к ней символ \0)

Файлы в работе программного обеспечения необходимы как места временного хранения информации или передачи информации пользователю

## Задание 24

24. Функции для работы с файлами WinAPI. Синхронный и асинхронный доступ к файлу.

За создание дескриптора файла отвечает функция Createfile

HANDLE CreateFile(

LPCTSTR lpFileName, // Указатель на имя файла (устройства)

DWORD dwDesiredAccess, //Параметры доступа

DWORD dwShareMode, //Разделяемый доступ

LPSECURITY\_ATTRIBUTES lpSecurityAttributes, //безопасность

DWORD dwCreationDistribution,// Описание

DWORD dwFlagsAndAttributes, // Атрибуты файла

HANDLE hTemplateFile // Файл шаблона

);

Запись данных в файл:

BOOL WriteFile(HANDLE hFile, //собственно указатель на файл

LPVOID lpBuffer, // указатель на буфер - откуда записываем данные в файл

DWORD nNumberOfBytesToWrite, //объем записываемых данных

LPDWORD lpNumberOfBytesWrite, //фактический размер записанных данных

LPOVERLAPPED lpOverlapped // флаг режима доступа к файлу: асинхронный(FILE\_FLAG\_OVERLAPPED)

//или синхронный(NULL)

конкретный пример записи в файл:

LPCSTR MyString = "Hello world";//буфер для записи (что записываем)

DWORD d = 0;

//функция записи в файл

WriteFile(hFile,//дескриптор открытого файла

MyString,//указываем буфер для записи

strlen(MyString), //указываем, сколько байт мы хотим записать

&d,//передаем указатель на DWORD

NULL//синхронный режим записи

);

Функция для чтения данных из файла:

За это отвечает функция ReadFile. Аргументы у неё такие же как его функции WriteFile,

Но при чтении есть некоторые особенности:

Надо указывать заведомо большой буфер для чтения (Так как мы не знаем Сколько информации находится в файле)

После прочтения строку нужно закрыть (Дописать к ней символ \0)

пример кода:

DWORD d = 0;//сколько фактически байт было прочитано

DWORD sizeBuffer = 521;//объем буфера

LPSTR str = malloc(sizeBuffer+1);//куда считывать

ReadFile(hFile, str, sizeBuffer, &d, NULL);

str[d] = '\0';

Пример со структурой Overlapped:

OVERLAPPED olf = { 0 }; //Структура, в которой задана позиция в файле

DWORD sizeBuffer = 512;//объем буфера

LPSTR str = malloc(sizeBuffer+1);//куда считывать

ReadFile(hFile, str, sizeBuffer, &d, &olf);

free(str);

olf.Offset = 0;//задаем смещение (позицию в файле)

LPSTR str1 = malloc(d + 1);

ReadFile(hFile, str1, d, &d1, &olf);

str1[d1] = '\0';

Имеется два типа синхронизации ввода - вывода (I/O) файлов: синхронный ввод - вывод (I/O) файла и асинхронный ввод - вывод (I/O) файла.

Асинхронный ввод - вывод (I/O) файла также называется, как перекрывающий ввод - вывод.

При синхронном вводе - выводе (synchronous file I/O) файла поток запускает операцию ввода/вывода (I/O) и немедленно вводит ждущееся состояние до тех пор, пока, запрос ввода-вывода не завершит работу. Поток, выполняющий асинхронный ввод - вывод (asynchronous file I/O) файла, отправляет запрос на ввод-вывод данных ядру. Если запрос принят ядром, поток продолжает обрабатывать другое задание до тех пор, пока ядро не подаст сигналы потоку, что операция ввода/вывода (I/O) полностью завершилась. Тогда поток прерывает работу со своим текущим заданием и обрабатывает данные от операции ввода/вывода (I/O) по мере необходимости. Асинхронные операции позволяют выполнять ресурсоемкие операции ввода-вывода без блокировки основного потока.

В ситуациях, когда ожидается запрос на ввод-вывод, который займет большое количество времени, такое как обновление или резервное копирование большой базы данных, асинхронный ввод - вывод (I/O) как правило - хороший способ оптимизировать эффективность обработки. Однако, для относительно быстрых операций ввода/вывода (I/O), непроизводительные издержки обработки запросов ядра на ввод-вывод и сигналов ядра могут сделать асинхронный ввод - вывод (I/O) менее выгодным, особенно если должны делаться много быстрых операций ввода/вывода (I/O). В этом случае, синхронный ввод - вывод (I/O) будет лучше.

Процесс открывает файл для асинхронного ввода - вывода (I/O) при его вызове в CreateFile, устанавливая флажок FILE\_FLAG\_OVERLAPPED в параметре dwFlagsAndAttributes. Если FILE\_FLAG\_OVERLAPPED не определяется, файл открыт для синхронного ввода - вывода (I/O). Когда файл был открыт для асинхронного ввода - вывода (I/O), указатель на структуру OVERLAPPED передается при вызове в ReadFile и WriteFile. Структура не передается при вызовах в ReadFile и WriteFile, выполняя синхронный ввод - вывод (I/O).

## Задание 25

25. Именованные каналы. Назначение и использование

Канал (англ. Pipe) - Область виртуального пространства, которое может быть использовано для совместного доступа различными процессами.

Однако в силу этого канал не может храниться отдельно от какого-то процесса (По факту дескриптор канала является глобальным указателем). Поэтому информация о нём будет очищена если завершить процесс, который его создал.

Будем называть процесс, Который создаёт канал с сервером, а процессы, которые подключаются к каналу - клиентами.

Каналы имеют несколько разновидностей:

• Симплексные или дуплексные

o Симплексный это однонаправленные. Например, сервер только записывает данные, а Клиенты только читают их. Или клиенты только пишут, а сервер только читает.

o Дуплексные это когда и клиент и сервер может и читать и писать

• Бинарные или текстовые. По аналогии с бинарными или текстовыми файлами

• С общим или разделяемым доступом к содержимому

• Именованные или анонимные

Именованные каналы полем дескриптора имеют также имя в виде строки. Причём, если дескриптор каждый раз разный, то имя является константой.

Имя является сетевым.

Сервер создаёт именованный канал с помощью функции CreateNamedPipe():

Она имеет следующую сигнатуру (мне кажется не обязательно это):

Сетевое имя канала

доступ (симплексный или дуплексный)

тип, режимы чтения и ожидания

максимальное число клиентов

размер выходного буфера, байты

размер входного буфера/байты

время паузы, миллисекунды (ожидание подключения)

структура безопасности

Проверить статус подключения клиента к серверу можно с помощью функции ConnectNamedPipe();

Для того чтобы установить подключение, на стороне клиента Должна быть вызвана функция SetNamedPipeHandleState();

Она имеет следующие аргументы:

• Дескриптор канала

• режим подключения

• максимальное количество пользователей

• Время ожидания

Пример:

BOOL isSuccess = SetNamedPipeHandleState(hNamedPipe,&dwMode,NULL,NULL);

Если клиент подключился к серверу, ConnectNamedPipe функция возвращает значение True иначе возвращает false;

Если сервер не отвечает, то функция SetNamedPipeHandleState возвращает FALSE.

Далее вся логика функционирования именного канала настраивается исходя из контекста задачи.

## //Задано Задание 26

26. Создание процессов. Функция запуска. Аргументы командной строки

Процесс – это любая запущенная программа. Создать процесс можно с помощью функции CreateProcess. Она имеет 10 полей, как правило, многим из них присваивается значение по умолчанию. При загрузке каждого процесса ему выделяются ресурсы

Функция запуска называется WinMain. Она принимает 4 параметра, отвечающих за дескриптор, совместимость с Win16, указатель на командную строку и значение, которые может быть передано в функцию Show Windows().

Аргументы командной строки – строковые аргументы, для обмена данными при создании нового процесса (входные данные для программы).

Когда мы запускаем исполняемый файл, мы создаем процесс. Вся информация о котором хранится в ОП. Передача данных между процессами очень важна. Например, по каким – либо критериям можно запускать системные утилиты. Функцией запуска является функция main, которая по хорошему счету должна что-то принимать и возвращать на этом и строится методика создания процессов. Если программа завершилась с кодом 1 (например, проверка скорости чтения диска), запускается и выполняется программа по очистке кэша и временных файлов, если 0 то программа не запускается.

Main принимает (int args, char\* argv[])

char\* argv[] – массив строк

int args – количество аргументов

Если через командную строку выполнить исполняемый файл и написать произвольную строку, можно получить эту строку в программе, если вывести argv[].