# **Теоретические вопросы:**

█ - не уверена, надо проверить и/или доделать

# 1. Windows HOOK. Назначение и особенности использования.

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

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

Она имеет следующие аргументы:

* Флаг, отвечающий за тип события
* Указатель на функцию, которая будет являться обработчиком события
* Дескриптор в библиотеке DLL, связанный с обработчиком события. Если обработчик находится не в dll, то ставим NULL
* Идентификатор потока, на которой распространяется хук. Если указать 0, то он будет работать со всеми потоками

Сама работа хука направлена на обработку системных событий. Поэтому также необходимо обрабатывать список входящих системных событий.

за это отвечает функция GetMessageW.

(более подробно: <https://habr.com/ru/company/icl_services/blog/324718/>)

# 2. Актуальность синхронизации потоков при параллельной обработке данных

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

Поток – последовательный набор инструкций, некий объект исполняемого процесса.

Существует несколько способов синхронизации потоков, однако все они в той или иной мере способствуют запрещению параллельного доступа к общему ресурсу.

Главным принципом является запрет параллельного доступа к общим ресурсам, например при помощи критической секции (это участок кода, который фиксируется для выполнения только одним потоком. Набор инструкций, который может выполняться только одним потоком одновременно).

# 3. Библиотеки DLL. Назначение и использование DLL-библиотек

По мере усложнения проекта количество используемых функций возрастает.

И не всегда использование многофайловых проектов решает эту проблему.

Помимо всего прочего использование множества функций в одном проекте влечет за собой следующие проблемы:

* Не все функции, описанные в проекте единовременно могут быть использованы, но они занимают пространство оперативной памяти и это сказывается на объёме исполняемого файла.
* Если одни и те же функции необходимо использовать в разных проектах, то код приходится дублировать. Причём не всегда копирование кода решает эту проблему.

Данные проблемы решаются за счет подключения динамических библиотек.

Динамические библиотеки подключаются к проекту и отключаются в тот момент времени, когда это надо. То есть достигается экономия оперативной памяти.

# 4. 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, поэтому информация о нём будет очищена если завершить процесс, который его создал.

Существуют:

1. Симплексный канал - это однонаправленные каналы. Например сервер только записывает данные, а Клиенты только читают их. Или клиенты только пишут, а сервер только читает.
2. Дуплексные каналы - это когда и клиент и сервер может и читать и писать.
3. Бинарные или текстовые. По аналогии с бинарными или текстовыми файлами
4. С общим или разделяемым доступом к содержимому.
5. Именованные или анонимные (использующиеся в конвейерах, — это на самом деле именованные каналы со случайным именем.)

Использование именованных каналов

Именованные каналы поем 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:

HANDLE CreateNamedPipe (

LPTSTR IpszPipeName, // строка с именем нового канала (сетевое)

DWORD fdwOpenMode, // доступ (симплексный или дуплексный)

DWORD fdwPipeMode, // тип, режимы чтения и ожидания

DWORD dwMaxInstances, // максимальное число клиентов

DWORD dwOutBuf, // размер выходного буфера, байты

DWORD dwInBuf, / размер входного буфера/байты

DWORD dwTimeout, // время паузы, миллисекунды (ожидание подключения)

LPSECURITY ATTRIBUTES Ipsa); // структура безопасности

Несмотря на все удобства, именованные каналы имеют ряд недостатков:

* Канал не может функционировать без сервера (Если программа сервер завершится, то канал закроется);
* Доступ к каналу нужно специально организовывать из программного кода.

Одним из решений данной проблемы является использование системного буфер обмена.

# 5. Линейный односвязный список. Особенности создания и примеры использования

Большинство данных в компьютере можно представить в виде списка из определенных структур.

Так как массивы имеют ряд недостатков - используют динамические структуры

Недостатки массивов 1.Необходимо выделять большое количество последовательно идущих секторов памяти.2.Это большое количество операций переприсваивания при при работе с элементами в середине массива.

ЛОС - Это самая простая из существующих динамических структур.

Каждый элемент списка является структурой, где часть полей отвечает за данные и одно поле отвечает за адрес такой же структуры, причем такое поле только одно. Данный подход удобен тем, что не нужно выделять последовательность ячеек памяти, то есть данные могут хранится в разных секторах памяти.

Создание списка:

пример использования односвязных списков:

#include <stdio.h>

struct MyStruct

{

int a;

struct MyStruct\* next;

};

typedef struct MyStruct s;

s\* create(int);

void show(s\*);

void delete(s\*);

s\* insert(s\*, s, int);

int main(void)

{

system("chcp 1251>nul");

s\* list1 = create(10);

show(list1);

s item = { 25,NULL };

list1 = insert(list1, item, 4);

show(list1);

delete(list1);

return 0;

}

s\* create(int n)

{

s\* start = malloc(sizeof(s));//создаем первый элемент

start->a = 1;

start->next = NULL;

s\* p, \* q;//указатели на предыдущий и следующий элемент ЛОС

p = start;

for (size\_t i = 0; i < n-1; i++)//создаем в цикле все остальные элементы

{

q = malloc(sizeof(s));//инициализируем следующий элемент

q->a = p->a + 1;

p->next = q;//поле с указателем предыдущего элемента содержит адрес следующего

p = q;//предыдущий элемент стал следующим

}

p->next = NULL;

return start;

}

void show(s\* list)

{

while (list)//пока list != NULL

{

printf("%d ", list->a);

list = list->next;

}

printf("\n");

}

void delete(s\* list)

{

s\* p = list;

while (p)

{

p = list->next;//запомнили следующий

free(list);//удалили предыдущий

list = p;//следующий стал первым

}

}

s\* insert(s\* list, s item, int k)

{

s\* el = malloc(sizeof(s));//выделяем память под новый элемент списка

el->a = item.a;//помещаем в него поле из структуры

if (k == 1)//если мы меняем первый элемент

{

el->next = list;

list = el;

}

else // если меняем не первый элемент

{

s\* start = list;//запоминаем голову списка

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

{

list = list->next;//сдвигаем на k позиций

if (!(list->next))

{

printf("вы ввели индекс, превышающий размеры списка. МЫ вставим элемент в конец списка\n");

break;

}

}

void\* temp = list->next;//обмен адресами

list->next = el;

el->next = temp;

list = start;//возвращаем голову на место

}

return list;

}

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

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

Пример:

#define N 10

int a[N];

int main(void)

{

system("chcp 1251>nul");

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

{

\*(a+i) = i \* i;

}

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

{

printf("%d ", \*(a+i));

}

return 0;

}

# 7. 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).

За работу процессов отвечают потоки. Поток рассматривается как последовательный набор инструкций, которая выполняется на процессоре. Каждый процесс имеет хотя бы один поток - основной поток.

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.

Процесс 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

Основной поток в процессе может также порождать и вспомогательные потоки.

Они нужны для параллельного выполнения операций. Желательно, чтобы данные, которые используют различные потоки не взаимодействовали между собой.

| Преимущества многопоточности | недостатки |
| --- | --- |
| 1. С использованием однопоточных процессов трудно организовать эффективное управление несколькими параллельно выполняющимися задачами, взаимодействующими между собой (ожидание и обработка пользовательского ввода) 2. Параллельной обработка файла с помощью потоков позволяет снизить общие накладные расходы системы. 3. Улучшенная реакция приложения - пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой. 4. Переключение между потоками требует от операционной системы гораздо меньше усилий, чем переключение между процессами. | 1. существует вероятность того, что один поток может случайно изменить данные, относящиеся к другому потоку. 2. может наблюдаться резкое ухудшение производительности 3. Трудная отладка многопоточной программы, поскольку взаимосвязи потоков очень трудно контролировать. |

# 8. Особенности использования строкового типа данных. Массивы строк

В [программировании](https://dic.academic.ru/dic.nsf/ruwiki/8183), строковый тип — [тип данных](https://dic.academic.ru/dic.nsf/ruwiki/1649675), значениями которого является произвольная последовательность символов, то есть строка. Каждая строковая переменная может быть представлена фиксированным количеством [байтов](https://dic.academic.ru/dic.nsf/ruwiki/4903) либо иметь произвольную длину.

В языке C в качестве строк используются обычные массивы символов.

Исторически сложилось два представления формата строк:

1. формат ANSI (устанавливает, что значением первой позиции в строке является ее длина, а затем следуют сами символы строки);

2. строки с завершающим нулем (используется в C - 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];

# 9. 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.

Настройка:

* Компилировать как код С
* Подсистема Windows
* Использование символов юникода
* Тип конфигурации динамическая библиотека.

Точка входа библиотеки DLL:

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;

}

Все экспортируемые из DLL функции должны иметь специальные соглашения о вызовах \_cdecl (соглашение о вызовах, используемое компиляторами для языка Си).

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.

Пример экспортируемой функции:

\_\_declspec(dllexport) int Hello(LPWSTR str);

int Hello(LPWSTR str)

{

MessageBox(NULL, str, L"Проверка связи", MB\_OK);

return 0;

}

# 10. 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.

Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си. Windows API использует stdcall (Соглашение о вызовах используется для вызова функций API Win32). Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в программах, исполняемых в среде Windows, могут пользоваться этим API(C++, C#, Pascal, Visual Basic).

Функция запуска приложений Windows

Функция называется WinMain. Принимает 4 параметра:

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 необходимо подключать заголовочный файл windows.h (создается системный процесс, который не имеет интерфейса).

Для ввода информации обычно используются файлы или другие способы (буфер обмена, именованный канал, файл)

Также вывод может осуществляться через MessageBox. Он принимает 4 параметра:

1. Дескриптор окна, который вызывает данный messagebox. Поскольку мы не работаем с оконными приложениями здесь ставим NULL

2. Непосредственно текст сообщения. Если для вывода используется текст в формате юникода, да то перед строкой ставятся префикс L

3. Заголовок окна messagebox . Это тоже может быть строка в формате ascii или Unicode

4. Набор параметров. Набор кнопок. Например кнопки OK отмена И другие. Также в качестве параметра можно вставить иконку

# 11. Особенности разработки программ для работы с системными событиями использования мыши и клавиатуры.

Для работы с системными событиями необходимо использовать хуки.  
Хуками называются перехватчики системных событий. Они позволяют программно подменять обработчик этих событий.

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

Сама работа хука направлена на обработку системных событий. Поэтому также необходимо обрабатывать список входящих системных событий.

За это отвечает функция GetMessageW. Она возвращает логическое значение, аргументы у неё следующие:

Для того чтобы интерпретировать события нажатия виртуальные клавиши используется функция TranslateMessage.

Для отправки сообщения в обработчик хука используется функция DispatchMessage.

Функция UnhookWindowsHookEx(hHook); Освобождает дескриптор хука.

Также для правильного интерпретации нажатые клавиши важно понятию регистр.

Символ должен быть напечатан в Верхнем регистре в двух раздельных случаях:

· Если зажата клавиша Shift

· если нажата клавиша caps lock

Для проверки этих ситуаций используются функция GetKeyState

Она принимает в качестве аргумента код виртуальной клавиши и возвращает значение типа SHORT.

И для того чтобы это значение перевести в логический тип используется операция конъюнкция с определенной константой

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

Первым подходом является использование блоков if и else, в которых можно описать какое-либо условие, которое может привести к ошибке работы программы и ее вылету, например, попытку деления на 0.  
Также можно в качестве возвращаемого значения метода или функции использовать enum или bool, которые позволят определить, когда произошла ошибка или что то пошло не так.

**\_try**, **\_except**, **\_finally**, **\_catch** и **\_throw**

Другим подходом могут являться директивы препроцессора:

* #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

Данные директивы позволяют исключить ошибки подключения каких-либо других директив или установить какое-либо условие для корректной работы программы.

# 13. Понятие “системное программирование”. предназначение и специфика системного программирования

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

Программы называется набор инструкций, понятных для конечного исполнителя.

Программный код - это зашифрованная версия программы (набор инструкций, написанный на языке, понятном постановщику задачи).

Транслятор называется средство, которое переводит инструкции с языка постановщика на язык исполнителя.

Компилятор переводит все инструкции сразу, А интерпретатор осуществляет построчный перевод.

Программирование - Это процесс написания программного кода.

Особенности работы программиста заключаются в специфическом стиле мышления, Который необходим для написания программ. 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.

# 14. Предназначение директив препроцессора и заголовочных файлов

Массивные программные продукты имеют достаточно большое количество строчек кода, чтобы избежать нагромождение кода, логически завершенные и повторно используемые фрагменты программного кода рекомендуется помещать в отдельные файлы и подключать его по мере необходимости.

Такие файлы называются файлами заголовков и имеют расширение .h, добавляется в проекте стандартным способом (с помощью #include).

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

Команды для препроцессора называются директивами(инструкции, записанные в тексте программы на СИ, и выполняемые до трансляции программы).

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

# 15. 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.

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).

Поток – последовательный набор инструкций, некий объект исполняемого процесса.

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

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

Обычно программа инициирует взаимодействие с ОС сами обращаются к операционной системе, посылая запросы. Однако Windows осуществляет обращение к прикладным программам. Программа находится в состоянии ожидания до тех пор, пока Windows не пошлет ей *сообщение* с помощью специальной функции, которая вызывается самой ОС. После того как сообщение будет принято, прикладная программа должна выполнить соответствующее действие. На принятое сообщение прикладная программа может вызвать одну или несколько функций API. По сравнению с остальными аспектами, именно механизм взаимодействия с Windows посредством сообщений больше всего определяет общий вид (структуру) всех Windows-программ.

Существует множество разнообразных типов сообщений, которые Windows может посылать программе. Например, когда выполняется щелчок кнопкой мыши в пределах окна прикладной программы, программе посылается сообщение о щелчке кнопкой мыши. Также посылаются сообщения каждый раз, когда пользователь нажимает клавишу, если происходит ввод информации.Поэтому построение прикладной программы должно исходить из предпосылки, что сообщения поступают к ней практически случайным образом. Вот почему Windows-приложения представляют собой управляемые прерываниями программы.

# 16. Принципы функционирования системного программного обеспечения

Систе́мное ПО — 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 прикладного программного обеспечения, 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.

Системное программное обеспечение, в отличие от прикладного, функционирует без непосредственного вмешательства пользователя, на уровне операционной системы. Такое ПО взаимодействует напрямую с ОС или еще ниже, с процессором и аппаратной составляющей ПК.

К системному ПО относятся утилиты, драйвера, программы, встроенные в аппаратное оборудование. Они, чаще всего, не взаимодействуют с пользователем и содержат инструкции, направленные к самой машине или ОС.

# 17. Реестр Windows. использование системного реестра программами

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.

Формально реестр представляет из себя базу данных.Элементами этой базы данных являются параметры. Организация параметров представляет из себя иерархию.Чаще всего реестр используется для хранения параметров операционной системы или прикладных программ.

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

Сам реестр имеет 5 глобальных веток:

* 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.

Остальные параметры имеют более длинные пути, которые начинаются в одной из глобальных веток.Крупные ветки называются ульями.Более мелкие ветки называются ключами реестра.Каждый ключ может содержать внутри себя другие ключи или параметры.

API функции для работы с реестром

Дескрипторов ключа реестра является переменная типа HKEY.HKEY Является указателем на соответствующую структуру.Для открытия ключа реестра используется функция RegOpenKey(HKEY\_CURRENT\_USER, NULL, &hKey);Она возвращает числовое значение (которое интерпретируется как код). Код успешного завершения этой функции определяется макросом ERROR\_SUCCESS

Функция RegCreateKey открывает ключ, если он есть или создаёт его. Имеет такие же параметры и тип возвращаемого значения, как и RegOpenKey. Для того, чтобы создать параметр используется функция RegSetValue.Возвращаемое значение у неё такое же, как и у предыдущих функций. Для того чтобы получить значение параметра Используется функция RegGetValue

# 18. Системный буфер обмена. особенности взаимодействия программного обеспечения с системным буфером обмена

Системный буфер обмена — это область памяти, предоставляемая операционной системой для временного хранения тех или иных данных. В зависимости от формата данных в буфере с ним могут взаимодействовать те или иные приложения. Доступ к нему может получить любая программа, запущенная на компьютере, а не только те приложения, между которыми вы переносите данные путем копирования и вставки.

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.

Примеры функций для работы с буфером:

int ClipboardInputText(LPWSTR buffer)//записать строку в системный буфер

{

DWORD len;//длина сообщения

HANDLE hMem;//дескриптор глобальной области памяти

len = wcslen(buffer) + 1; // определение длины строки в формате юникода

hMem = GlobalAlloc(GMEM\_MOVEABLE, len \* sizeof(LPWSTR)); //выделение памяти в глобальной области видимости

memcpy(GlobalLock(hMem), buffer, len \* sizeof(LPWSTR));// копирование области памяти из buffer в hMem

GlobalUnlock(hMem); //разблокировать содержимое этой памяти (сделать доступным для других программ)

OpenClipboard(0);//открыть буфер обмена

EmptyClipboard();//очистить буфер обмена

SetClipboardData(CF\_UNICODETEXT, hMem);//записать в буфер обмена данные соответствующего типа

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

return 0;

}

int ClipboardOutputText()//считать информацию из системного буфера

{

OpenClipboard(NULL);//открыть буфер обмена

LPWSTR Mess = (LPWSTR)GetClipboardData(CF\_UNICODETEXT);//Считать из глобального участка памяти, привести это все к строке

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

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

return 0;

}

# 19. Создание процессов. Функция запуска. Аргументы командной строки

Процесс – это любая запущенная программа. Создать процесс можно с помощью функции CreateProcess. Вся информация о котором хранится в ОП. При загрузке каждого процесса ему выделяются ресурсы.

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

Функцией запуска является функция main, которая по хорошему счету должна что то принимать и возвращать на этом и строится методика создания процессов.

Main принимать (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[].

# 20. Структура системного реестра. Типы данных в системном реестре.

Формально реестр представляет из себя базу данных.

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

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

Сам реестр имеет 5 глобальных веток:

* 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.

Остальные параметры имеют более длинные пути, которые начинаются в одной из глобальных веток.

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

# 21. Структуры данных. актуальность использования структур. передача структуры как параметра.

Структура — это, некое объединение различных переменных (даже с разными типами данных), которому можно присвоить имя.

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

struct MyStruct

{

int a;

char c;

};

int main(void)

{

system("chcp 1251>nul");

struct MyStruct m = { 10,'n' };

struct MyStruct\* p = &m;

printf("%d %c %d %c", m.a, m.c, (\*p).a, p->c);

return 0;}

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

# 22. Типы данных в ОС Windows

* Тип BYTE обозначает 8-разрядное беззнаковое символьное значение.
* Тип WORD — 16-разрядное беззнаковое короткое целое.
* Тип DWORD — беззнаковое длинное целое.
* Тип UINT — беззнаковое 32-разрядное целое.
* Тип LONG эквивалентен типу long.
* 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.

# 23. 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 = разрядности программы. Переменные однозначно идентифицируется двумя параметрами: Это её адрес (Номер 1 ячейки памяти) и Количество занимаемых ячеек.

# 24. Указатель на функцию. Назначение и примеры использования

В языке программирования C функция тоже 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

int (\*f)(int, int);//указатель на функцию с двумя аргументами типа int и возвращающую int

f = razn;

# 25. 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.

# 26. 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)

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';

Имеется два типа синхронизации ввода - вывода файлов:

Синхронный ввод - вывод файла и асинхронный ввод - вывод файла.

Асинхронный ввод - вывод файла также называется как перекрывающий ввод - вывод.

При синхронном вводе - выводе файла поток запускает операцию ввода/вывода и немедленно вводит ждущее состояние до тех пор, пока, запрос ввода-вывода не завершит работу.

Поток, выполняющий асинхронный ввод - вывод 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 полностью завершилась. Тогда поток прерывает работу со своим текущим заданием и обрабатывает данные от операции ввода/вывода по мере необходимости. Асинхронные операции позволяют выполнять ресурсоемкие операции ввода-вывода без блокировки основного потока.

В каких ситуациях лучше  
синхронный: для относительно быстрых операций ввода/вывода, так как затраты на обработки запросов ядра на ввод-вывод и сигналов ядра могут сделать асинхронный менее выгодным.

асинхронный: в ситуациях, когда ожидается запрос на ввод-вывод, который займет большое количество времени (обновление или резервное копирование большой базы данных).

**Практикос:**

# 1.запись и считывание строковых параметров в системный реестр

//Подсистема - консоль  
#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <strsafe.h>

#include <stdio.h>

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

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

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

return 1;

printf("Введите строку\n");

char str[50];

gets(str);

wchar\_t wtext[50];

mbstowcs(wtext, str, strlen(str) + 1);

LPWSTR ptr = wtext;

int length2 = snprintf(NULL, 0, "%s", str);

char\* Size2 = malloc(length2 + 1);

sprintf(Size2, "%s", str);

if (RegSetValueExW(hKey, L"Mykey", NULL, REG\_SZ, ptr, 22 \* sizeof(WCHAR)) == ERROR\_SUCCESS)

{

MessageBox(NULL, L"Ключу присвоено значение вашей строки", L"Информация", MB\_OK);

}

DWORD DataType = 0;

DWORD DataLen = 100500;

LPWSTR OutputString = malloc(512);

RegQueryValueExW(hKey, L"Mykey", NULL, &DataType, OutputString, &DataLen);

WideCharToMultiByte(CP\_ACP, WC\_COMPOSITECHECK, OutputString, -1, str, 0, NULL, NULL);

puts("Строка из реестра:");

puts(str);

}

# 2.запись и считывание числовых параметров в системный реестр

//Подсистема - Windows

#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;

}

# 

# 

# 3.использование системного буфера обмена для передачи строковых значений между процессами

//Подсистема - Windows

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <strsafe.h>

#include <stdio.h>

LPWSTR ClipboardOutputText();

int ClipboardInputText(LPWSTR);

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

{

while (TRUE)

{

LPSTR Data = ClipboardOutputText();

TCHAR Alert[] = L"Тсс!!\n";

TCHAR third[512];

swprintf(third, sizeof third, L"%s%s", Alert, Data);

if (\*Data != "")

{

MessageBoxW(NULL, &third, L"ВОР", MB\_OK | MB\_ICONINFORMATION);

}

Sleep(1000);

}

return 1;

}

int ClipboardInputText(LPWSTR buffer)//записать строку в системный буфер

{

DWORD len;

HANDLE hMenu;

len = wcslen(buffer) + 1;

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

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

GlobalUnlock(hMenu);

OpenClipboard(0);

EmptyClipboard();

SetClipboardData(CF\_UNICODETEXT, hMenu);

CloseClipboard();

return 0;

}

TCHAR\* ClipboardOutputText()//считать информацию из системного буфера

{

TCHAR\* Mess = NULL;

OpenClipboard(NULL);

HANDLE hClipboardData = GetClipboardData(CF\_UNICODETEXT);

Mess = (TCHAR\*)GlobalLock(hClipboardData);

GlobalUnlock(hClipboardData);

CloseClipboard();

EmptyClipboard();

return Mess;

}

# 4.использование условных блоков для анализа значений, возвращаемых системной функцией

//Подсистема - консоль  
#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");

}

}

# 5.обработка нажатия клавиши мыши в системе (выписать в messagebox какая клавиша нажата и сколько раз)

//Подсистема - Windows

#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);

}

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);

}

# 6. Отправка и получение данных из именованного канала не сложно //Подсистема - Консоль. Выполнять в 2 проектах.

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

}

}

# 7.получение кода клавиши в формате ASCII и использованием Windows HOOK

//Подсистема - Windows.  
#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))

# 8.получение названия клавиши с использованием функции GetKeyNameText (Windows HOOK)

//Подсистема - Windows.

#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";

}

}

# 

# 9.программа для записи строки в системный буфер обмена

//Подсистема - Windows.

#include <Windows.h>

#include <strsafe.h>

#include <stdio.h>

LPWSTR ClipboardOutputText();

int ClipboardInputText(LPWSTR);

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

{

while (TRUE)

{

LPSTR Data = ClipboardOutputText();

TCHAR Alert[] = L"Тсс!!\n";

TCHAR third[512];

swprintf(third, sizeof third, L"%s%s", Alert, Data);

if (\*Data != "")

{

MessageBoxW(NULL, &third, L"ВОР", MB\_OK | MB\_ICONINFORMATION);

}

Sleep(1000);

}

return 1;

}

int ClipboardInputText(LPWSTR buffer)//записать строку в системный буфер

{

DWORD len;

HANDLE hMenu;

len = wcslen(buffer) + 1;

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

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

GlobalUnlock(hMenu);

OpenClipboard(0);

EmptyClipboard();

SetClipboardData(CF\_UNICODETEXT, hMenu);

CloseClipboard();

return 0;

}

TCHAR\* ClipboardOutputText()//считать информацию из системного буфера

{

TCHAR\* Mess = NULL;

OpenClipboard(NULL);

HANDLE hClipboardData = GetClipboardData(CF\_UNICODETEXT);

Mess = (TCHAR\*)GlobalLock(hClipboardData);

GlobalUnlock(hClipboardData);

CloseClipboard();

EmptyClipboard();

return Mess;

}

# 10.программа для преобразования строки в число. Строка хранится в виде указателя LPWSTR

//Подсистема - Windows.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

int main()

{

LPWSTR \*str = L"10";

char buffer[500];

wcstombs(buffer, str, 500); // перевеод из LPWSTR в char[]

int a= atoi(buffer);

printf("%d", a);

return 0;

}

# 11.программа для преобразования числа в строку и склейки двух строк

//Подсистема - консоль

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char c1[256];

char c2[256];

int i,z;

printf("Введите первое число\n");

scanf("%d", &i);

printf("Введите второе число\n");

scanf("%d", &z);

sprintf(c1, "%d", i);

sprintf(c2, "%d", z);

char third[512];

snprintf(third, sizeof(third), "%s%s", c1, c2);

int d = 0;

while(third[d]!='\0')

{

printf("%c", third[d]);

d++;

}

return 0;

}

# 12.программа для считывания данных из системного буфера обмена

//Подсистема - Windows.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <strsafe.h>

#include <stdio.h>

LPWSTR ClipboardOutputText();

int ClipboardInputText(LPWSTR);

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

{

while (TRUE)

{

LPSTR Data = ClipboardOutputText();

TCHAR Alert[] = L"Тсс!!\n";

TCHAR third[512];

swprintf(third, sizeof third, L"%s%s", Alert, Data);

if (\*Data != "")

{

MessageBoxW(NULL, &third, L"ВОР", MB\_OK | MB\_ICONINFORMATION);

}

Sleep(1000);

}

return 1;

}

int ClipboardInputText(LPWSTR buffer)//записать строку в системный буфер

{

DWORD len;

HANDLE hMenu;

len = wcslen(buffer) + 1;

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

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

GlobalUnlock(hMenu);

OpenClipboard(0);

EmptyClipboard();

SetClipboardData(CF\_UNICODETEXT, hMenu);

CloseClipboard();

return 0;

}

TCHAR\* ClipboardOutputText()//считать информацию из системного буфера

{

TCHAR\* Mess = NULL;

OpenClipboard(NULL);

HANDLE hClipboardData = GetClipboardData(CF\_UNICODETEXT);

Mess = (TCHAR\*)GlobalLock(hClipboardData);

GlobalUnlock(hClipboardData);

CloseClipboard();

EmptyClipboard();

return Mess;

}

# 13.программа для считывания и записи строкового значения с использованием текстового файла(Компилировать как C, Подсистема Консоль)

#define \_CRT\_SECURE\_NO\_WARNINGS

#define ERROR\_OPEN\_FILE -3

#define INPUT\_FILE "D:\\Проекты\\ConsoleC\\Debug\\input.txt"

#include <Windows.h>

#include <stdio.h>

void main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

FILE\* inputFile;

inputFile = fopen(INPUT\_FILE, "r");

if (inputFile == NULL) {

printf("Error opening file %s", INPUT\_FILE);

exit(ERROR\_OPEN\_FILE);

}

char\* estr;

char str[50];

printf("Считаны строки : \n");

//Чтение (построчно) данных из файла в бесконечном цикле

while (1)

{

// Чтение одной строки из файла

estr = fgets(str, sizeof(str), inputFile);

//Проверка на конец файла или ошибку чтения

if (estr == NULL)

{

// Проверяем, что именно произошло: кончился файл

// или это ошибка чтения

if (feof(inputFile) != 0)

{

//Если файл закончился, выводим сообщение о завершении

//чтения и выходим из бесконечного цикла

printf("\nЧтение файла закончено\n");

break;

}

else

{

//Если при чтении произошла ошибка, выводим сообщение

//об ошибке и выходим из бесконечного цикла

printf("\nОшибка чтения из файла\n");

break;

}

}

//Если файл не закончился, и не было ошибки чтения

//выводим считанную строку на экран

printf("%s", str);

}

fclose(inputFile);

}

# 14.программа для считывания нескольких числовых значений из строки и из записи в числовой массив (разделитель - пробел) (Компилировать как C, подсистема Консоль)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("Введите строку\n");

char str[50];

int f;

char ch1[10], ch2[10], ch3[10];

float mass[3];

gets(str);

int d = 0, dd = 0;

int a = 0, b = 0, c = 0;

int check = 0;

while (str[d] != '\0')

{

if (str[d] != ' ')

{

if (str[d] == ',' || str[d] == '.' || str[d] == '-' || str[d] == '0' || str[d] == '1' || str[d] == '2' || str[d] == '3' || str[d] == '4' || str[d] == '5' || str[d] == '6' || str[d] == '7' || str[d] == '8' || str[d] == '9')

{

if (dd == 0)

{

ch1[a] = str[d];

a++;

}

if (dd == 1)

{

ch2[b] = str[d];

b++;

}

if (dd == 2)

{

ch3[c] = str[d];

c++;

}

d++;

}

else

{

check = 1;

d++;

}

}

else

{

dd++;

d++;

}

}

if (check == 1)

printf("Вы ввели некорректные данные");

else

{

mass[0] = atof(ch1);

mass[1] = atof(ch2);

mass[2] = atof(ch3);

for (int l = 0; l < 3; l++)

{

printf("Я число %f \n", mass[l]);

}

}

return 0;

}

Вариант 43п

#include <stdio.h>

#include <stdlib.h>//стандартная библиотека функций, она нужна для atoi

int main()

{

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

printf("Введите строку\n");

char str[10000];

gets(str);

//строка, содержащая в себе числа, но в формате символов

int k = 0;//переменная, которая будет хранить количество элементов числового массива

char\* Poisk = strtok(str, " ");//strtok позволяет разделить строку на отдельные "слова" по разделителю (у нас пробел). Poisk будет хранить указатель на первое "слово"

float Massive[100];//объявляем числовой массив, в который переведем числа из символов в реальные числа

while (Poisk != NULL)//пока не закончились отдельные "слова" в строке будем делать перево

{

Massive[k] = atof(Poisk);//atoi переводит символьную строку в число, если это получится. Поскольку у нас в строке написаны цифры, перевод получится без проблем. Записываем их в массив

k++;//прибавляем счетчик элементов массива

Poisk = strtok(NULL, " ");//переходим к следующему "слову" в строке

}

for (int i = 0; i < k; i++)//цикл будет выполнятся, пока мы не выведем все элементы массива (k, посчитали выше)

{

printf(" число %f\n", Massive[i]);//вывод на экран элемента массива

}

}

# 15.программа для считывания числового значения из файла (с сохранением его в числовом формате)(Создаем файл где угодно и копируем его путь, вставляем в INPUT\_FILE , компилировать как C, подсистема Консоль)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define ERROR\_OPEN\_FILE -3

#define INPUT\_FILE "D:\\Проекты\\ConsoleC\\Debug\\input.txt"

void main()

{

system("chcp 1251>nul");

float a;

FILE\* inputFile;

inputFile = fopen(INPUT\_FILE, "r");

if (inputFile == NULL) {

printf("Error opening file %s", INPUT\_FILE);

exit(ERROR\_OPEN\_FILE);

}

int i = fscanf(inputFile, "%f", &a);

if (i >= 1)

{

printf("В файле было %f",a);

}

else { printf("Не верное значение"); }

fclose(inputFile);

}

# 16.программа с импортом функции из файла DLL

**Файл вызова DLL**(Компилировать как C, Подсистема windows, не забудьте добавить ссылку!!)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#define PATHDLL L"DLL.dll"

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow)

{

HINSTANCE MyDLL;

if (!(MyDLL = LoadLibrary(PATHDLL))) return 1;//подключение DLL

MyFunction MyF1; //создали переменную типа указатель на вызываемую функцию

MyF1 = (MyFunction)GetProcAddress(MyDLL, "MyFunc");//инициализация указателя на функцию

MyF1(L"Привет, мир");

FreeLibrary(MyDLL);//отключение DLL

return 0;

}

**DLL (**Компилировать как C, Подсистема windows, Тип конфигурации Динамическая библиотека (.dll))

**:**

#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;

} \_\_declspec(dllimport) int MyFunc(LPWSTR);

int MyFunc(LPWSTR str)

{

MessageBox(NULL, str, L"Я сделал", MB\_OK);

return 0;

}

# 17.программа с использованием нескольких программных модулей (каждый модуль должен располагаться в отдельном файле .cpp) (Компилировать как C, создать 2 файла с расширением .c)

1 файл :

int main()

{

char\* strBuf = "Слово на вывод";

Vivod(strBuf);

}

2 файл:

#include <stdio.h>

extern int Vivod(char\* str)

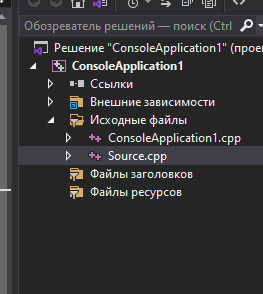
{

system("chcp 1251>nul");

printf("%s", str);

return 0;

}



# 

# 

# 18.программа с использованием пользовательских файлов заголовков(Компилировать как C, подсистема Консоль )

Код исходного файла:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "Header.h"

#include <Windows.h>

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int i, z;

printf("Введите первое число\n");

scanf("%d", &i);

printf("Введите второе число\n");

scanf("%d", &z);

int a = summ(i, z);

int b = mul(i, z);

printf("Сумма равна:%d, а произведение равно:%d", a, b);

return 0;

}

Код заголовочного:

#pragma once

int summ(int a, int b)

{

return a + b;

}

int mul(int a, int b)

{

return a \* b;

}

# 19.разработка программы для вычисления факториала в отдельном потоке(Компилировать как C, подсистема Консоль).

#include <Windows.h>

#include <stdio.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

DWORD WINAPI ThreadFactr(DWORD n);

int main()

{

system("chcp 1251>nul");

DWORD n;

printf("Введите число n ");

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

HANDLE hf[1];

hf[0] = CreateThread(NULL, 0, ThreadFactr, n, 0, 0);

WaitForMultipleObjects(1, hf, TRUE, INFINITE);

return 0;

}

DWORD WINAPI ThreadFactr(DWORD n)

{

int f = 1;

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

{

if (i == 0 || i == 1)

{

f = 1;

}

else

{

f \*= i;

}

printf("Факториал %d равен %d\n", i, f);

}

ExitThread(0);

}

# 20.разработка программы-секундомера. в одном потоке идет отсчет времени, а в другом осуществляется управление секундомером(Компилировать как C, подсистема Консоль)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <stdio.h>

//#include <malloc.h>

//#include <time.h>

VOID FF(VOID);

DWORD WINAPI Timer(BOOL[2]);

VOID ChangeTime(VOID);

DWORD WINAPI Sec();

SYSTEMTIME t;

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

{

system("chcp 1251");

system("cls");

FF();

return 0;

}

VOID ChangeTime(VOID)

{

system("cls");

DWORD d = 0;

printf\_s("Введите часы:");

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

t.wHour = d;

printf\_s("Введите минуты:");

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

t.wMinute = d;

printf\_s("Введите секунды:");

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

t.wSecond = d;

system("cls");

}

VOID FF(VOID)

{

BOOL F[2];

F[0] = TRUE;

F[1] = FALSE;

DWORD cmd = 0;

GetLocalTime(&t);

HANDLE Potok = CreateThread(NULL, 0, Timer, F, 0, 0);

while (1)

{

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

if (cmd == 1)

{

ResumeThread(Potok);

system("cls");

}

if (cmd == 0)

{

SuspendThread(Potok);

system("cls");

printf\_s("0-Остановка;1-Запуск;2-Сменить время;3-Поставить системное;4-Вкл/выкл секундомер,5-Вкл/выкл таймер");

}

if (cmd == 2)

{

SuspendThread(Potok);

ChangeTime();

ResumeThread(Potok);

}

if (cmd == 3)

{

GetLocalTime(&t);

system("cls");

ResumeThread(Potok);

}

if (cmd == 4)

{

F[1] = !F[1];

if (F[1])

{

t.wHour = 0;

t.wMinute = 0;

t.wSecond = 0;

t.wMilliseconds = 0;

}

else

{

GetLocalTime(&t);

}

system("cls");

}

if (cmd == 5)

{

F[0] = !F[0];

if (!F[0])

{

SuspendThread(Potok);

ChangeTime();

ResumeThread(Potok);

}

else

{

GetLocalTime(&t);

}

}

}

}

DWORD WINAPI Timer(BOOL F[2])

{

SYSTEMTIME st, nt;

GetLocalTime(&st);

GetLocalTime(&nt);

while (1)

{

GetLocalTime(&st);

if (F[1])

{

if ((st.wMilliseconds / 100) != (nt.wMilliseconds / 100) || nt.wHour != st.wHour || nt.wMinute != st.wMinute || nt.wSecond != st.wSecond)

{

system("cls");

nt = st;

printf\_s("%d:%d:%d:%d\n", t.wHour, t.wMinute, t.wSecond, t.wMilliseconds);

if (F[0])

{

t.wMilliseconds++;

if (t.wMilliseconds == 10)

{

t.wMilliseconds = 0;

t.wSecond++;

}

if (t.wSecond == 60)

{

t.wSecond = 0;

t.wMinute++;

}

if (t.wMinute == 60)

{

t.wMinute = 0;

t.wHour++;

}

if (t.wHour == 24)

{

t.wHour = 0;

}

}

}

}

else {

if (nt.wHour != st.wHour || nt.wMinute != st.wMinute || nt.wSecond != st.wSecond)

{

system("cls");

nt = st;

printf\_s("%d:%d:%d ", t.wHour, t.wMinute, t.wSecond);

if (F[0])

{

t.wSecond++;

if (t.wSecond == 60)

{

t.wSecond = 0;

t.wMinute++;

}

if (t.wMinute == 60)

{

t.wMinute = 0;

t.wHour++;

}

if (t.wHour == 24)

{

t.wHour = 0;

}

}

else

{

if (t.wHour == 0 && t.wMinute == 0 && t.wSecond == 0)

{

system("cls");

printf\_s("0:0:0 Время истекло");

}

else

{

if (t.wHour != 0 && t.wMinute == 0 && t.wSecond == 0)

{

t.wMinute = 59;

t.wSecond = 60;

t.wHour--;

}

if (t.wMinute != 0 && t.wSecond == 0)

{

t.wSecond = 60;

t.wMinute--;

}

if (t.wSecond != 0)

{

t.wSecond--;

}

}

}

}

}

}

ExitThread(0);

}

# 21.создание и настройка проекта WinAPI. вывод сообщения в messagebox(Компилировать как C, подсистема Windows)

#include <Windows.h>

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

int message = MessageBox(NULL, L"Нажмите на кнопку", L"Сделайте выбор", MB\_YESNOCANCEL);

switch (message)

{

case IDYES: {

MessageBox(NULL, L"Вы нажали Да", L"Выбор сделан", NULL);

break; }

case IDNO: {

MessageBox(NULL, L"Вы нажали Нет", L"Выбор сделан", NULL);

break; }

case IDCANCEL: {

MessageBox(NULL, L"Вы нажали Отмена", L"Выбор сделан", NULL);

break; }

default:

break;

}

}

# 22.создание и отладка файла DLL

//Вызов DLL Тип конфигурации -Приложение (exe.). Подсистема - Windows  
#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#define PATHDLL L"DLL.dll"

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow)

{

HINSTANCE MyDLL;

if (!(MyDLL = LoadLibrary(PATHDLL))) return 1;//подключение DLL

MyFunction MyF1; //создали переменную типа указатель на вызываемую функцию

MyF1 = (MyFunction)GetProcAddress(MyDLL, "MyFunc");//инициализация указателя на функцию

MyF1(L"Привет, мир");

FreeLibrary(MyDLL);//отключение DLL

return 0;

}

//DLL Тип конфигурации -Динамическая библиотека. Подсистема - Windows

#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;

}

\_\_declspec(dllimport) int MyFunc(LPWSTR str);

int MyFunc(LPWSTR str)

{

MessageBox(NULL, str, L"Я сделал", MB\_OK);

return 0;

}

# 23.создание консольной утилиты, которая печатает список переданных ей аргументов из командной строки(Открыть командную строку, скопировать путь exe файла программы, написать аргументы после пути и нажать энтер)

//Подсистема - консоль

//Пример D:\Проекты\ConsoleC\Debug\ConsoleC.exe 1 2 3

#include <stdio.h>

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

{

system("chcp 1251>nul");

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

{

printf("%s \n", argv[i]);

}

}

# 24.создание линейного односвязного списка из n элементов

#include <stdio.h>

struct MyStruct

{

int a;

struct MyStruct\* next;

};

typedef struct MyStruct s;

s\* create(int);

void show(s\*);

int main(void)

{

system("chcp 1251>nul");

s\* list1 = create(10);

show(list1);

s item = { 25,NULL };

return 0;

}

s\* create(int n)

{

s\* start = malloc(sizeof(s));//создаем первый элемент

start->a = 1;

start->next = NULL;

s\* p, \* q;//указатели на предыдущий и следующий элемент ЛОС

p = start;

for (size\_t i = 0; i < n - 1; i++)//создаем в цикле все остальные элементы

{

q = malloc(sizeof(s));//инициализируем следующий элемент

q->a = p->a + 1;

p->next = q;//поле с указателем предыдущего элемента содержит адрес следующего

p = q;//предыдущий элемент стал следующим

}

p->next = NULL;

return start;

}

void show(s\* list)

{

while (list)//пока list != NULL

{

printf("%d ", list->a);

list = list->next;

}

printf("\n");

}

# 25.создание программы с критической секцией(Компилировать C, консоль)

#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);

}

# 26.создание функции, аргументом которой является указатель на функцию(Компилировать как C, консоль)

#define \_CRT\_SECURE\_NO\_WARNINGS

#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;

}

# 3 задание:

https://github.com/Lolipopec/Ekz

# DLL:

#include "../header.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;

}

#pragma region FirstMod

LPWSTR Reading(LPWSTR path)

{

DWORD d = 0;

HINSTANCE FileHandle = CreateFile(PATH, GENERIC\_READ, FILE\_SHARE\_WRITE | FILE\_SHARE\_READ, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

LPWSTR ReadString = calloc(100500, sizeof(WCHAR));

ReadFile(FileHandle, ReadString, 100500 \* sizeof(WCHAR), &d, NULL);

CloseHandle(FileHandle);

ReadString[d / 2] = '\n';

ReadString[d / 2 + 1] = '\0';

return ReadString;

}

\_\_declspec(dllexport) U\* CreateUserArray(LPWSTR path, LPDWORD n)

{

LPWSTR str = Reading(path);

DWORD count = CountUsers(str);

U\* users = calloc(count, sizeof(U));

DWORD poz = 0, zap = 0, ind = 0;

LPWSTR strvozr = calloc(3, sizeof(TCHAR));

//i с одного для того что бы пропустить символ 65279 стоящий в начале файла

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

{

if (str[i] == '\n')

{

strvozr = calloc(3, sizeof(TCHAR));

zap++;

poz = 0;

ind = 0;

}

else

{

if (str[i] == ';')

{

poz++;

ind = 0;

}

else

{

if (poz == 0)

users[zap].f[ind] = str[i];

if (poz == 1)

users[zap].i[ind] = str[i];

if (poz == 2)

users[zap].o[ind] = str[i];

if (poz == 3)

{

if (str[i] == '\r')

{

strvozr[ind] = '\0';

users[zap].vozr = LpwstrToDword(strvozr);

free(strvozr);

}

else strvozr[ind] = str[i];

}

ind++;

}

}

}

\*n = count;

return users;

}

DWORD LpwstrToDword(LPWSTR str)

{

DWORD dw = 0;

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

{

dw += (str[i] - '0');

dw \*= 10;

}

return dw / 10;

}

DWORD CountUsers(LPWSTR str)

{

DWORD count = 0;

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

{

if (str[i] == L'\n') count++;

}

return count;

}

\_\_declspec(dllexport) DWORD WriteToCSV(U\* uses, LPDWORD count2)

{

DWORD w;

HINSTANCE FileHandle1 = CreateFile(L"SortUser.csv", GENERIC\_WRITE, FILE\_SHARE\_WRITE | FILE\_SHARE\_READ, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

TCHAR otvFile[512];

WriteFile(FileHandle1, uses[0].f, wcslen(uses[0].f) \* sizeof(WCHAR), &w, NULL);//Для записи символа начала строки 65279

for (size\_t i = 1; i < count2; i++)

{

uses[i].f[wcslen(uses[i].f)] = (WCHAR)';';

WriteFile(FileHandle1, uses[i].f, wcslen(uses[i].f) \* sizeof(WCHAR), &w, NULL);

uses[i].i[wcslen(uses[i].i)] = (WCHAR)';';

WriteFile(FileHandle1, uses[i].i, wcslen(uses[i].i) \* sizeof(WCHAR), &w, NULL);

uses[i].o[wcslen(uses[i].o)] = (WCHAR)';';

WriteFile(FileHandle1, uses[i].o, wcslen(uses[i].o) \* sizeof(WCHAR), &w, NULL);

swprintf(otvFile, 512, TEXT("%d\r\n"), uses[i].vozr);

DWORD len = 0;

while (otvFile[len] != '\0')

{

len++;

}

DWORD dz = 0;

WriteFile(FileHandle1,

otvFile,

len \* (sizeof(char) \* 2),

&dz,

NULL

);

}

CloseHandle(FileHandle1);

return 1;

}

#pragma endregion

#pragma region SecondMod

\_\_declspec(dllexport) U\* SortUserArray(U\* users, LPDWORD count, LPWSTR str)

{

DWORD ind = 0, indUser = 0, con = 0, f = 0;

char\* istr;

DWORD countF = wcslen(str);

for (DWORD i = 0; i < \*count; i++) // Поиск количества фамилий по условию для формирования новой структуры

{

istr = strstr(users[i].f, str);

if (istr != NULL)

{

con++;

}

}

U\* UsersSort = calloc(con + 1, sizeof(U));

UsersSort[0].f[0] = (WCHAR)65279;

indUser++;

for (DWORD i = 0; i < \*count; i++) //Поиск строк и записывание их

{

istr = strstr(users[i].f, str);

if (istr != NULL)

{

for (DWORD j = 0; j < wcslen(users[i].o); j++)

{

UsersSort[indUser].f[j] = users[i].f[j];

}

for (DWORD j = 0; j < wcslen(users[i].i); j++)

{

UsersSort[indUser].i[j] = users[i].i[j];

}

for (DWORD j = 0; j < wcslen(users[i].o); j++)

{

UsersSort[indUser].o[j] = users[i].o[j];

}

UsersSort[indUser].vozr = users[i].vozr;

indUser++;

}

}

\*count = con + 1;

return UsersSort;

}

#pragma endregion

#pragma region ThirdMod

\_\_declspec(dllexport) VOID AvgAge(U\* users, LPDWORD count)

{

DWORD d = count;

float avg = 0;

for (DWORD i = 0; i < count; i++)

{

avg += users[i].vozr;

}

avg = avg / (d-1);

char array[10];

sprintf(array, "%f", avg);

wchar\_t wc[10];

mbstowcs(wc, array, 10);

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;

LPWSTR ptr = wc;

int length2 = snprintf(NULL, 0, "%s", wc);

char\* Size2 = malloc(length2 + 1);

sprintf(Size2, "%s", wc);

if (RegSetValueExW(hKey, L"AvgAge", NULL, REG\_SZ, ptr, 22 \* sizeof(WCHAR)) == ERROR\_SUCCESS)

{

}

}

\_\_declspec(dllexport) VOID MaxLenF(U\* users, LPDWORD count)

{

DWORD MaxLen = 0,ind=0;

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

{

size\_t len1 = strlen(users[i].f);

size\_t len2 = strlen(users[i].i);

size\_t len3 = strlen(users[i].o);

if ((len1+ len2+ len3) > MaxLen)

{

MaxLen = len1 + len2 + len3;

ind = i;

}

}

char third[512];

swprintf(third, sizeof(third), L"%s;%s;%s", users[ind].f, users[ind].i, users[ind].o);

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;

LPWSTR ptr = third;

int length2 = snprintf(NULL, 0, "%s", ptr);

char\* Size2 = malloc(length2 + 1);

sprintf(Size2, "%s", ptr);

if (RegSetValueExW(hKey, L"MaxFIO", NULL, REG\_SZ, ptr,60\* sizeof(LPWSTR)) == ERROR\_SUCCESS)

{

}

}

#pragma endregion

#pragma region FourthMod

\_\_declspec(dllexport) VOID WriteToBuff(U\* uses, LPDWORD count)

{

int d = count;

wchar\_t buff[3500];

int t = 0;

for (int i = 1; i < d; i++)

{

for (int j = 0; j < strlen(uses[i].f); j++)

{

if (uses[i].f[j] != '\0')

{

buff[t] = uses[i].f[j];

t++;

}

}

buff[t] = ';';

t++;

for (int j = 0; j < strlen(uses[i].i); j++)

{

if (uses[i].i[j] != '\0')

{

buff[t] = uses[i].i[j];

t++;

}

}

buff[t] = ';';

t++;

for (int j = 0; j < strlen(uses[i].o); j++)

{

if (uses[i].o[j] != '\0')

{

buff[t] = uses[i].o[j];

t++;

}

}

buff[t] = ';';

t++;

int d = uses[i].vozr;

char array[3];

sprintf(array, "%d", d);

for (int j = 0; j < 3; j++)

{

if (array[j]!= '\0')

{

buff[t] = array[j];

t++;

}

}

buff[t] = '\r';

t++;

buff[t] = '\n';

t++;

}

buff[t] = '\0';

t++;

wchar\_t \*str = calloc(t,sizeof(wchar\_t));

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

{

str[i] = buff[i];

}

str[t + 1] = '\0';

LPWSTR buffer = str;

DWORD len;

HANDLE hMenu;

len = wcslen(buffer) + 1;

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

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

GlobalUnlock(hMenu);

OpenClipboard(0);

EmptyClipboard();

SetClipboardData(CF\_UNICODETEXT, hMenu);

CloseClipboard();

}

#pragma endregion

# header:

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <string.h>

#define PATH L"Users.csv"

struct User

{

WCHAR f[30];

WCHAR i[30];

WCHAR o[30];

DWORD vozr;

};

typedef struct User U;

# main:

#include "header.h"

#define PATHDLL L"EkzDLL.dll"

typedef U\* (\_cdecl\* MyFunction)(LPWSTR, LPDWORD);

typedef U\* (\_cdecl\* MyFunction1)(U\*, LPDWORD, LPWSTR);

typedef U\* (\_cdecl\* MyFunction2)(LPWSTR, LPDWORD);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)

{

HINSTANCE MyDLL;

if (!(MyDLL = LoadLibrary(PATHDLL))) return 1;

MyFunction UserList = (MyFunction)GetProcAddress(MyDLL, "CreateUserArray");

MyFunction1 UserSort = (MyFunction1)GetProcAddress(MyDLL, "SortUserArray");

MyFunction2 WriteToCSV = (MyFunction2)GetProcAddress(MyDLL, "WriteToCSV");

MyFunction MaxLenF = (MyFunction)GetProcAddress(MyDLL, "MaxLenF");

MyFunction AvgAge = (MyFunction)GetProcAddress(MyDLL, "AvgAge");

MyFunction WriteToBuff = (MyFunction)GetProcAddress(MyDLL, "WriteToBuff");

DWORD countUser = 0, countSortUser;

U\* users = UserList(PATH, &countUser);

LPWSTR str = L"Иванов";

countSortUser = countUser;

U\* uses = UserSort(users, &countSortUser, str);

WriteToCSV(uses, countSortUser);

countSortUser = countUser;

uses = UserSort(users, &countSortUser, str);

MaxLenF(users, countUser);

AvgAge(users, countUser);

WriteToBuff(users, countUser-1);

return 0;

}

3 задание на шарпе

<https://github.com/Drenbb/Ekz>

Main:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using dll;

using Microsoft.Win32;

namespace Ekz

{

internal class Program

{

[STAThread]

static void Main(string[] args)

{

string s="";

using (StreamReader sr = new StreamReader(File.Open("Users.csv", FileMode.OpenOrCreate)))

{

while (sr.Peek() > -1)

{

s = sr.ReadToEnd();

}

}

Class1.read(s);

Class1.Second();

Class1.Third();

Class1.fourth();

RegistryKey registryKey = Registry.CurrentUser;

MessageBox.Show(registryKey.GetValue("AverageAge") + " " + registryKey.GetValue("MaxLen"));

Console.ReadKey();

}

}

}

Dll:

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace dll

{

public struct FIO//Структура для хранения данных из CSV(ФИО и возраст пользователей)

{

public string Imya;

public string Familia;

public string Otch;

public int Age;

public FIO(string Familia, string Imya, string Otch, int Age)

{

this.Imya = Imya;

this.Familia = Familia;

this.Otch = Otch;

this.Age = Age;

}

public string Show()

{

string s = " Фамилия -" + Familia + "Имя- " + Imya + " Отчество- " + Otch + ", Возраст - " + Age + "\n";

return s;

}

}

public static class Class1

{

public static FIO[] p;

public static void read(string s)//Метод перевода из переменной стринг в массив стринг

{

int z = 0;

p = new FIO[1];

string[] ss = s.Split(';', '\n', '\r');//Удаление не нужных сиволов

try

{

for (int i = 0; i < ss.Length; i++)//Запись из массива стринг в массив структуры

{

p[z].Familia = ss[i];

p[z].Imya = ss[i + 1];

p[z].Otch = ss[i + 2];

p[z].Age = Convert.ToInt32(ss[i + 3]);

z++;

i += 4;

Array.Resize(ref p, p.Length + 1);

}

}

catch

{

Array.Resize(ref p, p.Length - 1);

}

}

public static void Second()//Метод поиска пользователей с фамилией Иванов

{

using (StreamWriter sw = new StreamWriter(File.Open("Ivanovy.csv", FileMode.Create)))

{

foreach (FIO f in p)

{

if (f.Familia.Contains("Иванов"))

{

sw.Write(f.Familia + ";" + f.Imya + ";" + f.Otch + ";" + f.Age + "\r\n");

}

}

}

}

public static void Third()//Метод поиска среднего возраста всех пользователей и самого Длинного ФИО

{

float summ = 0;

for (int i = 0; i < p.Length; i++)

{

summ += p[i].Age;

}

float average = summ / p.Length;

using (StreamWriter sw = new StreamWriter(File.Open("AverageAge.csv", FileMode.Create)))

{

sw.Write("Средний возраст всех пользователей " + average);

}

int d = p[0].Familia.Length + p[0].Imya.Length + p[0].Otch.Length;

int ind = 0;

for (int i = 0; i < p.Length; i++)

{

if (d < p[i].Familia.Length + p[i].Imya.Length + p[i].Otch.Length)

{

ind = i;

d = p[i].Familia.Length + p[i].Imya.Length + p[i].Otch.Length;

}

}

string f = p[ind].Familia + " " + p[ind].Imya + " " + p[ind].Otch;

int z = f.Length;

RegistryKey registryKey = Registry.CurrentUser;

if (registryKey.GetValueNames().Contains("MaxLen"))//Запись самого длинного ФИО в реестр

{

if (f != registryKey.GetValue("MaxLen"))

{

registryKey.DeleteValue("MaxLen");

registryKey.SetValue("MaxLen", f);

}

}

else

{

registryKey.SetValue("MaxLen", f);

}

if (registryKey.GetValueNames().Contains("AverageAge"))//Запись среднего возраста в реестр

{

if (f != registryKey.GetValue("AverageAge"))

{

registryKey.DeleteValue("AverageAge");

registryKey.SetValue("AverageAge", average);

}

}

else

{

registryKey.SetValue("AverageAge", average);

}

}

public static void fourth()//Сохранение всего списка в Буффер обмена

{

string k = "";

foreach (FIO f in p)

{

k += f.Familia + ";" + f.Imya + ";" + f.Otch + ";" + f.Age + "\r\n";

}

Clipboard.SetText(k);

}

}

}

# Теория 43П:

**1. Windows HOOK. Назначение и особенности использования.**

Хуками называются перехватчики системных событий. Они позволяют программно подменять обработчик этих событий (накладывать свои функции вместо системных).

Сама работа ХУКА направлена на получение системных событий. Поэтому необходимо обрабатывать список входящих системных событий.

Для того чтобы интерпретировать события нажатия виртуальные клавиши используется функция TranslateMessage (оно принимает событие нажатия на клавишу, например).

Она возвращает логическое значение и принимает в качестве аргумента указатель на структуру MSG (то есть системное сообщение, событие).

За перевод системного события отвечает TranslateMessage. Она принимает указатель на структуру MSG (системное событие), а возвращает числовой код или символ, который уже можно использовать в программе.

**2. Актуальность синхронизации потоков при параллельной обработке данных**

В случае, когда два или более потоков пытаются получить одновременный доступ к какому-либо общему ресурсу (например, к данным в оперативной памяти), поведение программы может быть неверным. Значение может быть записано раньше времени или прочитана раньше времени (так как два потока получают доступ к области памяти в рандомное время)

Существует несколько способов синхронизации потоков, однако все они в той или иной мере способствуют запрещению параллельного доступа к общему ресурсу.

Есть способ синхронизации «Критическая секция».

Критическая секция - это участок кода, который фиксируется для выполнения только одним потоком (то есть может выполняться только одним потоком одновременно). Пока критическая секция занята одним потоком, другие ждут своей очереди и не взаимодействуют с ним.

**3. Библиотеки DLL. Назначение и использование DLL-библиотек**

По мере усложнения проекта количество используемых функций возрастает, и не всегда использование Многофайловых проектов решает эту проблему. Использование множества функций в одном проекте влечет за собой следующие проблемы:

· Не все функции используются одновременно, но они компилируются все вместе, чем занимают память. Они не нужны в данный момент, но все равно занимают место, как, например, игрушки в телефоне, в которые ты редко заходишь).

· Если одни и те же функции необходимо использовать в разных проектах, то код приходится копировать. Это неудобно и занимает больше времени, чем использование длл

Динамические библиотеки подключаются к проекту и отключаются в тот момент времени, когда это надо. Таким образом достигается экономия оперативной памяти.

Сами динамические библиотеки содержать в себе те же функции и код, который можно использовать. Туда разумно помещать функции, которые используются часто в разных проектах. Или же туда можно запихивать функции, которые используются редко, чтобы они не занимали место в оперативной памяти при работе программы.

При подключении длл библиотек нужно

1. Подключить DLL. (Создается дескриптор данной библиотеки)

2. Импортируется функция из этой библиотеки

3. Выполнить эту функцию в коде

4. Отключить библиотеку (освободить память дескриптора библеотеки)

Нужно обязательно указать, какие функции будут экспортироваться из длл (выносится в проект), а в основной программе указать, какие функции нужно импортировать из длл (вытащить оттуда). Причем, нужно, чтобы это совпадало: если импортируешь функцию в программу, нужно, чтобы в длл она была указана, как экспортируемая.

Кстати, можно подключить библиотеку WinAPI к программе на языке C#. Это ускоряет время выполнение самой программы, но средства языка C# не смогу анализировать функции библиотеки и предотвращать проблемы.

**4. Именованные каналы. Назначение и использование**

Канал (англ. Pipe) - область виртуального пространства, которое может быть использовано для совместного доступа различными процессами (своего рода виртуальный файл, в который разные программы могут записывать данные и читать их, обмениваться этими данными)

Однако канал не может храниться отдельно от какого-то процесса. Поэтому информация о нём будет очищена если завершить процесс (или программу), который его создал.

Будем называть процесс, который создаёт канал Сервером, а процессы, которые подключаются к каналу - Клиентами.

Каналы имеют несколько разновидностей:

· Симплексные - это однонаправленные. Например, Cервер только записывает данные, а Клиенты только читают их. Или клиенты только пишут, а сервер только читает.

· Дуплексные - это когда и Клиенты, и Сервер могут 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 помимо дескриптора имеют также Имя в виде строки. Причём это имя должно всегда оставаться одним и тем же, как у Сервера, так и у Клиента

Имя является сетевым (доступно другим компьютерам, что позволяет обменивать данными не только разным процессам, но и, по сути, разным машинам).

Простое использование именованного канала – обмен сообщениями (простой мессенджер). Сервер и клиент могут писать друг другу сообщения в виде строк, читать их и отвечать (если канал работает в дуплексном режиме).

Остальные способы использования зависят от контекста: можно передавать и получать системные данные, тот же кейлоггер сделать, обмениваться результатами вычислений и т.п.

**5. Линейный односвязный список. Особенности создания и примеры использования**

Линейный односвязный список – это более удобный (для компьютера) массив. По сути, это набор структур: в первом поле содержится какое-то значение, во втором содержится указатель на следующий элемент. В отличии от обычных массивов, односвязный список не записывается в память подряд, элементы списка могут находится в разных областях памяти. Еще одно преимущество – список можно дополнять, достаточно просто записать еще одно новое число, а предыдущему дать указатель на это новое число.

Пример: числовой односвязный список. Создаем структуру с двумя полями: первое поле – числовое, второе поле – указатель. В первое поле будем записывать число, во второе – указатель на следующий элемент.

[1, указатель на второй элемент] -> [2, указатель на третий элемент] -> и т.д.

Кстати, у односвязных списков есть и свой недостаток: чтобы получить доступ, например, к 256 элементу, нужно пройтись по всем предыдущим, так как путь к нужному элементу известен только тому, что стоит перед ним (в нашем случае 255 элементу).

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

Фактически массив является указателем на последовательность элементов определенного типа. Имя массива является указателем на его первый элемент. А объём памяти равен суммарному объему всех элементов с учетом их типа. Чтобы выделить память под массив, необходимо найти область в оперативной памяти, которая будет равна объему массива (то есть все элементы хранятся в памяти в одной области, в отличие от односвязного списка).

По сути, массив – это множество элементов одного типа. Доступ к каждому элементу можно получить не перебирая все предыдущие элементы (как с односвязным списком), достаточно указать его номер (например, Massive[35] – это 35 элемент массива). У массивов фиксированная длина, которая задается во время определения массива, эта длина постоянна, и менять ее нельзя.

**7. Многопоточный режим работы компьютера. Преимущества использования многопоточности**

Многопоточность — свойство платформы (например, операционной системы) или приложения, состоящее в том, что процесс, созданный в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть каждый со своим временем, без строгих рамок.

По-настоящему многопоточный режим может достигаться только на много процессорных системах, так как только несколько процессоров могут выполнять задачи параллельно. На однопроцессорных же системах применяется псевдопаралелльный метод – потоки все равно выполняются друг за другом, но делятся на более мелкие куски, что позволяет, например, двум потокам выполнятся как бы параллельно. Это не особо сокращает общее время выполнение обоих процессов, но позволяет им выполнятся в одном темпе, что в некоторых случаях может оказаться полезным (например, если мониторить(следить) несколько показателей в реальном времени).

Преимущество многопоточного режима в одном процессе – выполнение нескольких потоков одновременно, параллельно. Несколько потоков будут выполнятся одновременно, что сократит общее время выполнения процесса.

**8. Особенности использования строкового типа данных. Массивы строк**

Строковые типы данных необходимы в первую очередь пользователю, нежели человеку. Строка – это массив символов, последним элементом которого является закрывающий ноль.

В основном строки используются для диалога с пользователем, так как машине все равно на эти наши буквы).

Для создания массива строк используется двумерный массив символов. Левый индекс определяет число строк, а правый индекс - максимальное число символов в каждой строке. Доступ к отдельным строкам очень прост - необходимо просто определить левый индекс (то есть, номер строки).

**9. 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 во на запуск как динамическую библиотеку.

Проект DLL имеет некоторые особенности.

Он настраивается также, как и проект WinAPI, но помимо этого ещё необходимо указать в настройках проекта на вкладке «Общие» -> «Тип конфигурации» -> «Динамическая библиотека», а в на вкладке «C/C++» -> «Дополнительно» -> «»

Кстати, можно подключить библиотеку WinAPI к программе на языке C#. Это ускоряет время выполнение самой программы, но средства языка C# не смогу анализировать функции библиотеки и предотвращать проблемы.

Точка входа у библиотеки DLL тоже своя:

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;

}

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 импортировать или экспортировать ее низя.

Для импорта функции необходимо сначала создать указатель на функцию с сигнатурой, Которую мы хотим вызвать. Также мы должны указать соглашение о вызовах. (\_cdecl).

За подключение библиотеки отвечает функция LoadLibrary (ей передается путь к длл), за отключение отвечает FreeLibrary (принимает дескриптор длл).

За инициализацию функции из DLL отвечает функция GetProcAddress. Она принимает два параметра: дескриптор DLL и строку, содержащую имя импортируемой функции. Возвращает указатель на эту функцию.

**10. Особенности разработки программ в WinAPI**

Операционная система Windows может предоставлять интерфейс для выполнения определённых системных задач сторонними программами. Такой интерфейс называется WinAPI.

WinAPI Представляет из себя набор функций, структур и различных параметров.

Функция запуска Windows приложений называется WinMain(аргументы: дескриптор запущенного приложения, дескриптор win16(неактуален), указатель на командную строку, значение для вывода командной строки).

Перед запуском приложений Windows необходимо особым образом настроить проект: в качестве подсистемы проекта выбрать Windows и выбрать компилятор Си. Также при работе с API системы Windows необходимо подключать заголовочный файл Windows.h.

В данном случае создается системный процесс, который не имеет интерфейса. Для ввода информации обычно используются файлы или другие способы (буфер обмена, именованный канал и т.п.). Вывод информации также осуществляется в файл или другой источник данных. Также вывод может осуществляться в диалоговое окно вывода, за это отвечает функция MessageBox(аргументы: дескриптор окна(мы ставим NULL), текст сообщения, заголовок окна, параметры(кнопки, иконки и т.п.)).

**11. Особенности разработки программ для работы с системными событиями использования мыши и клавиатуры.**

Для работы с системными событиями необходимо использовать ХУКИ. Они перехватывают системные сообщения. Так можно перехватывать нажатия на клавиши или использование мыши.

Для клавы: HHOOK hHookKeyboard = SetWindowsHookExW(WH\_KEYBOARD\_LL, LogKey, NULL, NULL);

WH\_KEYBOARD\_LL – это обработка нажатий на клавиши клавы, LogKey – функция, которая будет вызываться при нажатии на клавишу. Остальное – значения по умолчанию, поэтому NULL.

Для того чтобы интерпретировать события нажатия виртуальные клавиши используется функция TranslateMessage (оно принимает событие нажатия на клавишу, например).

Она возвращает логическое значение и принимает в качестве аргумента указатель на структуру MSG (то есть системное сообщение, событие).

Для отправки сообщения в обработчик ХУКА используется функция DispatchMessage.

Она возвращает числовой код, а в качестве аргумента также принимает указатель на структуру MSG.

Также для правильного интерпретации нажатые клавиши важно понять регистр.

Символ может быть напечатан в Верхнем регистре в двух раздельных случаях:

• Если зажата клавиша Shift

• Если нажата клавиша CapsLock

Для проверки этих ситуаций используются функция GetKeyState (она позволяет получить регистр, который потом можно преобразовать в логическое значение кнопка заглавная или Незаглавная, например).

Для перевода английской клавиша в русскую можно воспользоваться самостоятельно написанной функцией через switch (если нажата клавиша ‘q’, русской будет ‘й’ ну и так со всеми).

Полученные данные можно записать в файл, чтобы узнать, какие клавиши были нажаты пользователем. По сути, это простой кейлоггер.

Все это можно делать как с клавишами клавиатуры, так и с кнопками мыши.

Для мыши потребуется сделать свой ХУК:

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

Он будет перехватывать нажатие на кнопку мыши, или поворот колесика.

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

Достаточно распространённый и простой обработать ошибки – проверить их в условном блоке. Например, многие функции возвращают логические значения, которые можно легко проверить в условии. Другие функции в случае правильного выполнения возвращают 0, иначе другие значения, что также можно записать в условие оператора if.

Такая же ситуация обстоит и с указателями. При обращении к ним, возвращается правильное значение, если все правильно, или 0, если что-то пошло не так. Собственно, достаточно просто проверить, если при обращении к указателю возвращается ноль, значит есть ошибка при чтении памяти.

**13. Понятие “системное программирование”. предназначение и специфика системного программирования**

Системное программирование – это создание набора инструкций на языке, который будет понятен как программисту, так и системе, для которой предназначается инструкция. Системой является ОС компьютера, а программист в данном случае создает программы и утилиты, направленные на работу в ОС.

Чаще всего под системными программами понимаются те, что взаимодействуют напрямую с процессором. Эти программы направлены не на пользователя (как прикладные), а на аппаратную составляющую. Системное программирование предназначено для создания программ, которые будут взаимодействовать напрямую с машиной, выполнять различные задачи на низком уровне, облегчая (так скажем) жизнь компьютеру. Системными программами являются и драйвера, и утилиты для работы с аппаратами (например, утилита для дефрагментации диска), более комплексные программы. Все это объединяет то, что они не взаимодействуют с пользователем, максимум – интерфейс для задачи начальных параметров, все остальное будет выполнятся в ОС и не потребует вмешательства пользователя. Исходя из всего это, делаем вывод, что программист должен быть хорошо осведомлен в сфере аппаратного обеспечения, и непосредственно языка системного программирования (наш – это Си).

**14. Предназначение директив препроцессора и заголовочных файлов**

Препроцессор - это специальная программа, которая осуществляет алгоритмические действия перед компиляцией основного кода. Команды для препроцессора называются директивами. Директивы начинаются с символа «#», в конце не ставится «;».

Список основных директив:

· #include. Вставляет содержимое из текстового файла (заголовочного) в то место, где прописана эта команда. Короче, позволяет использовать какие-то функции из заголовочных файлов в проге.

· #define. Имеет три основных применения:

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

o Задание констант. Используется в качестве альтернативы глобальным переменным (постоянные значения, которые ну никак не изменишь)

o Задание макроопределений (макросов) (заранее заданные команды или условия, маленькие функции)

· #undef. Отменяют задание параметра (противоположность define)

· #ifdef - Условие компиляции, если определённый параметр задан (если есть какой-то define)

· #ifndef - Условие компиляции, если определённый параметр не задан (если какого-то define нет)

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

· #elif - Директивы для создания вложенных условий (внутри основного #if)

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

· #error - Внесение искусственной ошибки для компиляции (для теста, например)

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

Windows осуществляет обращение к прикладным программам. Этот процесс происходит примерно следующим образом:

1. Программа находится в ожидании

2. ОС отправляет сообщение программе на запуск

3. Программа запускается

4. Программа выполняет действия

Подробнее.

Программа находится в состоянии ожидания до тех пор, пока ОС не пошлет ей сообщение.

Сообщение передается прикладной программе посредством специальной функции, которая вызывается самой ОС.

После того как сообщение будет принято, предполагается, что прикладная программа должна выполнить соответствующее действие. Запускается процесс программы – некая абстрактность, задача, выполняемая процессором, которой необходимы ресурсы для своей работы. Процесс делится на потоки – условная единица, которая выполняется в процессоре. Каждый поток выполняется поочередно, по завершению выполнения всех потоков, завершается и выполнение процесса, а соответственно и выполнении программы. Каждый процесс на протяжении всего жизненного цикла имеет дескриптор – такой описатель, который содержит номер процесса, область памяти, где размещен сегмент кода, приоритет процесса, данные о его состоянии.

**16. Принципы функционирования системного программного обеспечения**

Системное программное обеспечение, в отличие от прикладного, функционирует без непосредственного вмешательства пользователя, на уровне операционной системы. Такое ПО взаимодействует напрямую с ОС или еще ниже, с процессором и аппаратной составляющей ПК.

К системному ПО относят утилиты, драйвера, программы, встроенные в аппаратное оборудование. Они, чаще всего, не взаимодействуют с пользователем и содержат инструкции, направленные к самой машине или ОС.

**17. Реестр Windows. Использование системного реестра программами**

База данных, которая хранит в себе параметры и настройки большинства программ, включая системные. Реестр представляет собой иерархическую структуру. Включает пять основных ключей, которые, в свою очередь, содержат в себе другие ключи и т.д. Все ключи имеют некоторые параметры разных типов (числовые, строковые или логические), которые и являются параметры программ.

Главная цель реестра – хранить параметры всех программ в одном месте, чтобы при надобности, программы могли получать к ним доступ, читать или изменять.

**18. Системный буфер обмена. Особенности взаимодействия программного обеспечения с системным буфером обмена.**

Системный буфер обмена — это область памяти, предоставляемая операционной системой для временного хранения тех или иных данных. Доступ к нему может получить любая программа. Каждая программа с помощью простых функций и интерфейсов может получить доступ к буферу обмена, просмотреть данные, которые в нем хранятся, захапать их себе, если вдруг они потребуются, или записать свои данные, которые смогут прочитать другие программы.

Буфер обмена – это простой способ приложениям обмениваться данными, не записывая их в файлы.

**19. Создание процессов. Функция запуска. Аргументы командной строки**

**20. Структура системного реестра. Типы данных в системном реестре.**

Всего в реестре Windows есть пять основных ветвей:

· HKEY\_CLASSES\_ROOT, (раздел для типов файлов, которые нужны винде)

· HKEY\_CURRENT\_USER, (инфа о пользователе, который зашел в данный момент)

· HKEY\_LOCAL\_MACHINE, (инфа об оборудовании ПК)

· HKEY\_USERS, (инфа обо всех пользователях, которые зареганы на этом пк)

· HKEY\_CURRENT\_CONFIG. (настройка оборудования для текущего сеанса использования пк)

Каждая ветвь содержит множество различных ключей, отвечающих за группы программ и сами программы. Ключи, в свою очередь, содержат параметры программ разных типов, таких как:

· REG\_BINARY - Двоичные данные

· REG\_DWORD - Число

· REG\_QWORD - 64-разрядное числовое значение

· REG\_EXPAND\_SZ - Текст и переменные

· REG\_FULL\_RESOURCE\_DESCRIPTOR - Идентификатор ресурса устройства

· REG\_LINK - Путь к файлу

· REG\_MULTI\_SZ - Массив строк

· REG\_NONE - Зашифрованные данные

· REG\_RESOURCE\_LIST - Список ресурсов устройств

· REG\_RESOURCE\_REQUIREMENTS\_LIST - Идентификатор ресурса устройства

· REG\_SZ - Текст

**21. Структуры данных. Актуальность использования структур. Передача структуры как параметра.**

Большинство данных в компьютере можно представить в виде списка из определенных структур.

Причём хранение осуществляется именно в виде списков, а не в виде массивов, поскольку массивы имеют ряд недостатков:

· Большее количество последовательно идущих секторов памяти

· Сложность редактирования или удаления элементов в середине массива

Для того чтобы избежать эти недостатки и актуальнее использовать динамические структуры данных.

Чтобы передать структуру можно либо указать ее глобально, либо передавать указатель на ее первый элемент (он же указатель на указатель структуры).

**22. Типы данных в ОС Windows**

· Тип BYTE обозначает 8-разрядное беззнаковое символьное значение.

· Тип WORD — 16-разрядное беззнаковое короткое целое. (положительное число)

· Тип DWORD — беззнаковое длинное целое. (большое положительное число)

· Тип UINT — беззнаковое 32-разрядное целое. (тоже большое положительное число)

· Тип LONG эквивалентен типу long. (большое число, может быть отрицательным)

· 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 (описателя в котором хранится инфа о процессе, например его адрес в памяти, состояние, приоритет).

**23. Указатель как тип данных. Особенности работы с указателями**

Указатель – переменная, содержащая адрес того или иного значения в памяти. Он используется для того, чтобы получать доступ не просто к переменной, которая хранит значение, а к самому значению прямиком в памяти. Это будет полезно, если одно значение нужно менять в разных функциях программы. Поскольку указатель – это тоже переменная, на нее также можно сделать указатель, таким образом получится указатель на указатель (а можно еще парочку добавить и будет ПИЗДОС).

Массив также является указателем на последовательность элементов. Двумерный массив – это указатель на последовательность одномерных массивов. То есть, указатель двумерного массива – это номера строк, каждая из которых тоже является последовательностью значений.

24. Указатель на функцию. Назначение и примеры использования

Поскольку функция тоже занимает место в памяти, на нее также можно сделать указатель. В таком случае, при обращении к указателю на функцию будет вызываться сама эта функция. Выглядеть это будет так:

int razn(int a, int b)

{

return a - b;

}

int main(void)

{

int (\*f)(int, int);//указатель на функцию razn (она выше) с двумя аргументами типа int и возвращающую int

f = razn;//присваиваем указателю адрес функции выше

printf("%d", f(2, 3));//выводим на экран получившуюся разность (мы вызвали ее, используя указатель)

return 0;

}

Это может пригодится, когда необходимо напрямую обращаться к памяти, где хранится функция.

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

Под файловым типом данных (файлом) понимается упорядоченная совокупность любого количества компонент одного типа. Понятие файла носит широкий характер. В качестве файла может выступать обычный файл, который хранится на жестком диске, либо коммуникационный порт компьютера, устройство печати, клавиатура и т.д.

Чаще всего, файлы используются для хранения или обмена данными между разными программами. Если на файл не наложены ограничения, каждая программа может открыть его, просмотреть данные, записать свои, или удалить их.

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

За создание дескриптора файла отвечает функция Createfile

HANDLE CreateFile(указатель на имя файла (устройства), параметры доступа (можно ли читать и/или писать), разделяемый доступ (можно ли вместе читать и/или писать), безопасность (NULL по умолчанию), описание открытия (открыть готовый, создать новый и т.п.), атрибуты файла (FILE\_ATTRIBUTE\_NORMAL по умолчанию), файл шаблона (NULL если он не нужен));

За запись отвечает функция WriteFile

BOOL WriteFile(указатель на файл, указатель на буфер (откуда записываем данные в файл), объем записываемых данных (ограничение по записи), фактический размер записанных данных (может быть меньше ограничения), флаг режима доступа к файлу: асинхронный(FILE\_FLAG\_OVERLAPPED) или синхронный(NULL)).

За чтение из файла 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)

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

DWORD sizeBuffer = 521;//объем буфера для чтения (ограничение)

LPSTR str = malloc(sizeBuffer+1); //строка для считывания из файла

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

str[d] = '\0'; //закрываем строку, добавляя в ее конец /0

При асинхронном доступе к файлу процесс продолжит выполнение, даже если чтение или запись в файл еще не закончилась. Синхронный – наоборот, соответственно, процесс продолжит выполнение после того, как взаимодействие с файлом закончится.