**Нулевой блок вопросов**

1. Win32 API

Для использования данного API в языке c++ подключаются библиотеки Windows.h, conio.h, process.h. В них определено множество деректив #define, необходимых для работы с API (напр. переопределены базовые типы данных)

1. Потоком в Windows называется объект ядра, которому операционная система выделяет процессорное время для выполнения приложения. Каждому потоку принадлежат следующие ресурсы:

* код исполняемой функции;
* набор регистров процессора;
* стек для работы приложения;
* стек для работы операционной системы;
* блок окружения, который содержит служебную информацию для работы потока.

Разделяют рабочие потоки (working threads) – потоки, с которыми может работать пользователь, и потоки интерфейса пользователя (user interface threads) – потоки, работающие без участия пользователя.

Для создания потоков используются функции:

HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes, // атрибуты защиты

DWORD dwStackSize, // размер стека потока в байтах

LPTHREAD\_START\_ROUTINE lpStartAddress, // адрес исполняемой функции

LPVOID lpParameter, // адрес параметра

DWORD dwCreationFlags, // флаги создания потока

LPDWORD lpThreadId // идентификатор потока

);

uintptr\_t \_beginthreadex(

void \*security,

unsigned stack\_size,

unsigned ( \_\_stdcall \*start\_address )( void \* ),

void \*arglist,

unsigned initflag,

unsigned \*thrdaddr

);

Примеры использования (лаб. 2):

HANDLE hThread;

DWORD IDThread;

auto param = new Array(a, n);

hThread = (HANDLE)\_beginthreadex(NULL, 0, &Min\_Max, (void\*)param, 0, (unsigned int\*)&IDThread);

if (hThread == NULL)

return GetLastError();

//hThread = CreateThread(nullptr, 0, Min\_Max, (LPVOID)param, 0, &IDThread); //another way to open thread

WaitForSingleObject(hThread, INFINITE);

CloseHandle(hThread);

1. Мьютекс – объект ядра, который находится в сигнальном состоянии, если он не принадлежит ни одному потоку. В противном случае мьютекс находится в несигнальном состоянии. Одновременно мьютекс может принадлежать только одному потоку.

Создается мьютекс вызовом функции CreateMutex, которая имеет следующий прототип:

HANDLE CreateMutex(

LPSECURITY\_ATTRIBUTES lpMutexAttributes, // атрибуты защиты

BOOL bInitialOwner, // начальный владелец мьютекса

LPCTSTR lpName // имя мьютекса

);Мьютекс захватывается потоком посредством любой функции ожидания, а освобождается функцией ReleaseMutex, которая имеет следующий прототип:

BOOL ReleaseMutex(

HANDLE hMutex // дескриптор мьютекса

);

Для того

чтобы получить доступ к уже созданному мьютексу, поток может также использовать функцию OpenMutex,

которая имеет следующий прототип:

HANDLE OpenMutex(

DWORD dwDesiredAccess, // доступ к мьютексу

BOOL bInheritHandle // свойство наследования

LPCTSTR lpName // имя мьютекса

);

К сожалению, на данный момент мне приходилось в лабораторных работать с mutex.

1. Событием называется оповещение о некотором выполненном действии. В программировании события используются для оповещения одного потока о том, что другой поток выполнил некоторое действие.

В операционных системах Windows события описываются объектами ядра Events. При этом различают два типа событий:

* события с ручным сбросом;
* события с автоматическим сбросом.

Различие между этими типами событий заключается в том, что событие с ручным сбросом можно перевести в несигнальное состояние только посредством вызова функции ResetEvent, а событие с автоматическим сбросом переходит в несигнальное состояние как при помощи функции ResetEvent, так и при помощи функции ожидания. При этом отметим, что если события с автоматическим сбросом ждут несколько потоков, используя

функцию WaitForSingleObject, то из состояния ожидания освобождается только один из этих потоков.

Создаются события вызовом функции CreateEvent, которая имеет следующий прототип:

HANDLE CreateEvent(

LPSECURITY\_ATTRIBUTES lpSecurityAttributes, // атрибуты защиты

BOOL bManualReset, // тип события

BOOL bInitialState, // начальное состояние события

LPCTSTR lpName // имя события

);

Примеры использования (лаб. 3):

HANDLE st = CreateEvent(NULL, TRUE, FALSE, NULL);

SetEvent(st);

struct numsThread

{

int\* arr;

int num;

HANDLE stop = CreateEvent(NULL, TRUE, FALSE, NULL);

HANDLE\* event = new HANDLE[2];

};

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

{

if (!check[i])

{

ResetEvent(arrF[i].stop);

SetEvent(arrF[i].event[1]); //continue the process

quit = true;

}

}

//закрытие

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

{

CloseHandle(hThread[i]);

CloseHandle(stop[i]);

CloseHandle(arrF[i].event[0]);

CloseHandle(arrF[i].event[1]);

}

**Второй блок вопросов**

1. ООП декомпозиция - это процесс разбиения большой системы на более мелкие и управляемые компоненты, которые могут быть легко разработаны, тестированы и поддерживаемы. Декомпозиция также может включать разделение функциональности на классы и методы, чтобы облегчить понимание и использование кода.
2. Статический полиморфизм - это способность языка программирования решать, какой метод вызывать на основе типа аргументов, переданных в метод, во время компиляции. Это означает, что при статическом полиморфизме компилятор знает, какой метод вызывать до выполнения программы.
3. Инкапсуляция - это концепция объектно-ориентированного программирования, которая позволяет скрыть детали реализации объекта от пользователя, предоставляя только интерфейс для работы с ним. Это означает, что пользователь не знает, как объект управляет своими данными и функциональностью, но может использовать его методы для выполнения задач. Инкапсуляция позволяет создавать более безопасный и модульный код, который может быть легко изменен без влияния на другие части программы.

**Третий блок вопросов**

1. Builder Pattern - это паттерн проектирования, который позволяет создавать сложные объекты пошагово. Он используется для создания объектов, которые могут иметь различные конфигурации.

Основная идея Builder Pattern заключается в том, что мы создаем отдельный класс-строитель (Builder), который отвечает за создание объекта. Этот класс имеет методы для установки значений свойств объекта, которые мы хотим создать. Затем мы вызываем метод build() для создания объекта.

Пример использования Builder Pattern на практике может быть в создании заказа в интернет-магазине. Класс OrderBuilder будет отвечать за создание объекта Order. Методы этого класса будут устанавливать свойства заказа, такие как товары, адрес доставки, способ оплаты и т.д. Затем мы вызываем метод build() для создания объекта Order.

Еще один пример использования Builder Pattern может быть в создании SQL-запросов. Класс QueryBuilder будет отвечать за создание SQL-запроса. Методы этого класса будут устанавливать свойства запроса, такие как таблицы, условия, сортировка и т.д. Затем мы вызываем метод build() для создания SQL-запроса.

Использование Builder Pattern позволяет инкапсулировать процесс создания сложных объектов, что делает код более читаемым и поддерживаемым. Также это позволяет создавать объекты с различными конфигурациями без необходимости создания множества конструкторов.

1. Decorator Pattern - это паттерн проектирования, который позволяет добавлять новые функциональности к объектам без изменения их исходного кода. Он используется для расширения функциональности объекта во время выполнения программы.

Основная идея Decorator Pattern заключается в том, что мы создаем отдельный класс-декоратор (Decorator), который содержит ссылку на объект, который мы хотим декорировать. Этот класс имеет те же методы, что и декорируемый объект, но может добавлять дополнительные функции. Затем мы создаем другой класс-декоратор, который также содержит ссылку на объект, и так далее.

Пример использования Decorator Pattern на практике может быть в создании приложения для обработки изображений. Класс Image будет отвечать за загрузку и отображение изображения. Классы-декораторы, такие как BlackAndWhiteFilter и SepiaFilter, будут добавлять дополнительные функции обработки изображения. Например, BlackAndWhiteFilter может преобразовывать изображение в черно-белое, а SepiaFilter может добавлять эффект старины.

Еще один пример использования Decorator Pattern может быть в создании приложения для обработки текстовых документов. Класс Document будет отвечать за загрузку и отображение текстового документа. Классы-декораторы, такие как BoldDecorator и ItalicDecorator, будут добавлять дополнительные функции форматирования текста. Например, BoldDecorator может выделять текст жирным шрифтом, а ItalicDecorator - курсивом.

Использование Decorator Pattern позволяет добавлять новые функциональности к объектам без изменения их исходного кода, что делает код более гибким и поддерживаемым. Также это позволяет создавать множество комбинаций декораторов для объекта, что дает возможность создавать уникальные функциональности.