ВАРИАНТ 1

# Нулевая группа вопросов

1. **Приведите Win API, необходимое для решения Лабораторной работы номер 3**

// Создание события

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

HANDLE hContinueEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

// Создание потока

HANDLE hThread = CreateThread(NULL, 0, MarkerThread, threadParam, 0, NULL);

// Ожидание события

WaitForSingleObject(hStartEvent, INFINITE);

// Установка события

SetEvent(hContinueEvent);

// Критическая секция

CRITICAL\_SECTION cs;

InitializeCriticalSection(&cs);

EnterCriticalSection(&cs);

// работа с общими данными

LeaveCriticalSection(&cs);

1. **Что такое поток в ОС Windows;**

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.

Все эти ресурсы образуют контекст потока в Windows. В Windows различаются потоки двух типов:

* рабочие потоки (working threads);
* потоки интерфейса пользователя (user interface threads).

Рабочие потоки выполняют различные фоновые задачи в приложении. Потоки интерфейса пользователя

связаны с окнами и выполняют обработку сообщений, поступающих этим окнам.

Каждое приложение имеет, по крайней мере, один поток, который называется первичным (primary) или

главным (main) потоком. В консольных приложениях это поток, который исполняет функцию main. В

приложениях с графическим интерфейсом это поток, который исполняет функцию WinMain.

1. **Что такое Мьютекс;**

Для решения проблемы взаимного исключения между параллельными потоками, выполняющимися в контексте разных процессов, в операционных системах Windows используется объект ядра мьютекс. Слово мьютекс является переводом английского слова mutex, которое в свою очередь является сокращением от

выражения mutual exclusion, что на русском языке значит взаимное исключение. Мьютекс 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

);

1. **Что такое Событие (объект синхронизации);**

Событием называется оповещение о некотором выполненном действии. В программировании события используются для оповещения одного потока о том, что другой поток выполнил некоторое действие. Сама же задача оповещения одного потока о некотором действии, которое совершил другой поток, называется задачей условной синхронизации или иногда задачей оповещения.

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

);

1. **Сравнительный анализ стандарта C++ 98 и любого более свежего стандарта (в контексте лабораторных)**

| **Аспект** | **C++98** | **C++11/14/17/20** |
| --- | --- | --- |
| **Потоки** | Нет (WinAPI/POSIX) | <thread>, <mutex>, <atomic> |
| **Умные указатели** | Только std::auto\_ptr (устарел) | unique\_ptr, shared\_ptr, weak\_ptr |
| **STL** | Базовые контейнеры | unordered\_map, move-семантика, лямбды |
| **Функциональность** | Функторы | Лямбды, std::function |
| **Типы** | Явное указание | auto, decltype |

# Общие вопросы

1. **Что такое ООП? – полное определение;**

Объектно-ориентированное программирование (ООП) — это парадигма программирования, основанная на концепции объектов, которые объединяют данные (поля, атрибуты) и методы (функции) для работы с этими данными.

4 основных принципа ООП:

1. Инкапсуляция – сокрытие внутренней реализации объекта (например, private-поля в классе).
2. Наследование – создание новых классов на основе существующих (class Derived : public Base).
3. Полиморфизм – возможность объектов с одинаковым интерфейсом вести себя по-разному (виртуальные функции, перегрузка операторов).
4. Абстракция – выделение ключевых характеристик объекта, игнорируя несущественные детали (интерфейсы, абстрактные классы).
5. **Магическое число 7 Миллера? – привести не менее 7 примеров из IT;**

Закон Миллера утверждает, что человек может одновременно удерживать в оперативной памяти 7 ± 2 элемента.

7 примеров из IT:

1. Лимит вкладок в браузере – большинство пользователей держат открытыми около 7 вкладок.
2. Максимальная вложенность меню (например, в Windows – не более 7 уровней).
3. Количество пунктов в навигационном меню сайта (обычно 5–9).
4. Число параметров функции (рекомендуется не больше 7, иначе сложно запомнить порядок).
5. Количество колонок в таблицах (если больше 7–9, восприятие ухудшается).
6. Число элементов в CLI-интерфейсе (например, git имеет около 7 основных команд верхнего уровня).
7. Количество строк в commit-сообщении (рекомендуется 50–72 символа – примерно 7–9 слов).
8. **Энтропия ПО? – привести не менее 5 примеров нег энтропийных мер при разработке ПО;**

Энтропия ПО – мера хаоса и беспорядка в коде, ведущая к усложнению поддержки.

5 способов борьбы с энтропией:

1. Рефакторинг – устранение "кодового запаха" (например, разбиение god-object на мелкие классы).
2. Документация – комментарии, README, Swagger (как в REST API из pet-проекта).
3. Юнит-тесты – автоматическая проверка корректности (например, тесты для CriticalSection в лабораторной).
4. CI/CD – автоматизация сборки и деплоя (GitHub Actions, GitLab CI).
5. Паттерны проектирования – Singleton для логгера, Factory для создания потоков.
6. **5 признаков сложной системы по Гради Бучу – по каждому признаку привести не менее 2-х пояснительных примером из Ваших Лабораторных работ или pet проектов;**

Согласно Гради Бучу, сложная система обладает:

1. Иерархичностью

* Пример 1: В лабораторной с потоками есть иерархия: main → marker потоки.
* Пример 2: В pet-проекте с микросервисами – API-шлюз → сервисы → БД.

1. Наличием подсистем

* Пример 1: В ОС потоки → процессы → диспетчер памяти.
* Пример 2: В игре – рендеринг, физика, AI.

1. Слабой связанностью

* Пример 1: Потоки marker работают независимо, общаясь через Event.
* Пример 2: REST API с разделением на auth-service и data-service.

1. Эмерджентностью (свойства системы ≠ сумме свойств частей)

* Пример 1: Deadlock возникает только при взаимодействии потоков.
* Пример 2: Баг в многопоточной программе, которого нет в однопоточной версии.

1. Динамическим поведением

* Пример 1: Потоки marker могут блокироваться и возобновляться.
* Пример 2: Алгоритм балансировки нагрузки в кластере.

1. **Закон иерархических компенсаций Седова – привести не менее 5 исторических примеров применения закона Седова в эволюции IT сферы;**
2. Переход от монолитов к микросервисам

Монолит сложен → микросервисы просты, но их много.

1. Отказ от ASM в пользу языков высокого уровня

Ассемблер сложен → Python прост, но требует интерпретатора.

1. Виртуализация (Docker, Kubernetes)

Настройка серверов сложна → контейнеры просты, но оркестратор усложняется.

1. SQL → NoSQL

Реляционные БД сложны для масштабирования → MongoDB проще, но теряет ACID.

1. REST → GraphQL

REST требует множества эндпоинтов → GraphQL имеет один endpoint, но сложный парсер запросов.