

Модуль "Многопоточное программирование". Вопросы.

1. Потоки

a. Параллелизм и конкурентность.

Что такое параллелизм и что такое конкурентность? Что такое процесс и что такое поток? Организация параллелизма с использованием процессов и с использованием потоков. В чём преимущества и недостатки этих подходов.

b. Потоки. Класс `std::thread`

Что такое поток? Создание нового потока в языке C++ с использованием объекта класса `std::thread`. Методы `join` и `detach`. Что произойдёт если выбросится исключение (в новом потоке, или в потоке, который создаёт новый поток)? Передача аргументов в функцию потока.

c. Возврат данных из функции потока

Возврат данных из потока с использованием глобальной переменной. Класс `std::reference_wrapper`. Функция `std::ref`. Чем похожи и чем отличаются объекты класса `std::reference_wrapper` и обычные ссылки? Возврат данных из потока с использованием объектов типа `std::reference_wrapper`. Почему нельзя для возврата данных из потока использовать обычные ссылки?

d. Другое

Идентификация потоков. Передача владения потоком с использованием семантики перемещения. Создание произвольного количества потоков. Использование стандартных контейнеров и стандартных алгоритмов, для работы с произвольным количеством потоков. Функция `std::mem_fn`.

2. Мьютексы

a. Состояние гонки.

Что такое разделяемые данные? Что такое состояние гонки (race condition)? Проблематичные и безобидные состояния гонки. Что такое гонка данных (data race) и к чему она приводит? При каких условиях возникает гонка данных в программе, написанной на языке C++ и к чему она приводит?

b. Стандартный мьютекс

Защита разделяемых данных с помощью мьютекса. Класс `std::mutex`. Блокировка и разблокировка мьютекса. Методы `lock`, `unlock` и `try_lock`.

c. Стандартные классы `lock_guard` и `unique_lock`

В чём недостатки класса `std::mutex`? Класс `std::lock_guard`. В чём преимущество `std::lock_guard` перед `std::mutex`? Класс `std::unique_lock`. В чём преимущества и недостатки `std::unique_lock` перед `std::lock_guard`?

d. Взаимоблокировка

Взаимоблокировка (deadlock). Решение проблемы взаимоблокировки с помощью стандартной функции `std::lock`.

e. `call_once`

3. Механизмы синхронизации

a. Условные переменные

Условные переменные. Класс `std::condition_variable` и как им пользоваться? Методы `wait`, `notify_one` и `notify_all`. Ложные пробуждения (spurious wake).

b. Запуск асинхронной задачи

Запуск асинхронной задачи с помощью функции `std::async`. Возврат значения из асинхронной задачи с помощью объекта класса `std::future`.

c. Класс `packaged_task`

Класс задачи – `std::packaged_task`. Зачем могут понадобиться объекты класса `std::packaged_task`? Методы класса `std::packaged_task`: `get_future`, `operator()`. Передача объекта класса `std::packaged_task` в другие функции и потоки.

d. Класс `promise`

Класс `std::promise`. Методы класса `std::promise`: `get_future`, `set_value` и `set_exception`.

4. Потокобезопасные стек и очередь с блокировками

a. Потокобезопасные структуры данных

Что такое потокобезопасная структура данных? Что такое потокобезопасная структура данных с блокировками? Написание своего потокобезопасного стека с блокировками? Являются ли стандартные контейнеры STL потокобезопасными?

b. Недостатки стандартного класса `std::stack`

Стандартный класс `std::stack` и его методы `push`, `top` и `pop`. В чём недостатки этого класса и интерфейса для работы этим классом? Почему в стандартной библиотеке языка C++ стек реализован так, как он реализован?

c. Потокобезопасный стек с блокировками

Реализация потокобезопасного стека на основе класса `std::stack`. Реализация методов `push` и `pop` такого стека. Безопасность относительно исключений для такого стека.

d. Потокобезопасная очередь с блокировками

Потокобезопасная очередь с блокировками. Реализация методов `push`, `try_pop` (в случае пустой очереди возвращает `false`) и `wait_and_pop` (в случае пустой очереди ожидает пока в очередь не добавится ещё один элемент). Безопасность относительно исключений для такой очереди.

e. Потокобезопасная очередь с блокировками на основе односвязного списка

Использование двух мьютексов для защиты головы и хвоста очереди. Реализация такой очереди и её методов. Безопасность относительно исключений для такой очереди.

5. Модели памяти

a. Барьеры памяти

Причина неопределённого поведения при гонке данных. Когерентность кэша. Примеры кода, когда процессор эффективно может поменять местами исполнение инструкций. Барьеры памяти. `LoadLoad`, `LoadStore`, `StoreLoad`, `StoreStore` барьеры. `Acquire` и `release` барьеры.

b. Модели памяти в языке C++

Упорядочение доступа к памяти. Упорядочения `memory_order_seq_cst`, `memory_order_acquire`, `memory_order_release` и `memory_order_relaxed`. Функция `std::atomic_thread_fence`.

c. Атомарные типы и операции над ними.

Атомарные переменные. В чём отличие атомарных переменных от обычных переменных? Класс `atomic_flag` и его методы `clear` и `test_and_set`. Атомарные типы `atomic<T>` и методы `load`, `store` и `compare_exchange`. Реализация спинлока (простейшего мьютекса) на основе атомарной переменной.

6. Потокобезопасные стек и очередь без блокировок

a. Основные определения

Неблокирующие структуры данных. Структуры данных, свободные от блокировок. Структуры данных, свободные от ожидания.

b. Реализация потокобезопасного стека без блокировок

Реализация потокобезопасного стека без блокировок (без устранения утечек памяти).

c. Управление памятью в структурах данных без блокировок.

Метод подсчёта количества потоков, выполняющих `pop`. Метод указателей опасности (`hazard pointers`). В чём преимущества и недостатки каждого из методов. Реализация потокобезопасной очереди без блокировок (с устранением утечек памяти).

7. Пул потоков.

Что такое пул потоков? Реализация пула потоков на языке C++. Ожидание задачи, переданной пулу потоков. Предотвращение конкуренции за очередь работ. Занимание работ.