Модуль 18

- Базовая многопоточность
- Обзор классов std::thread, std::future
- Способы синхронизации
- Обзор std::mutex, std::atomic

Базовая многопоточность. Процесс vs поток.

Процесс — экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы (например, процессорное время и память). Каждый процесс выполняется в отдельном адресном пространстве: один процесс не может получить доступ к переменным и структурам данных другого. Если процесс хочет получить доступ к чужим ресурсам, необходимо использовать межпроцессное взаимодействие. Это могут быть конвейеры, файлы, каналы связи между компьютерами и многое другое.
Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. У каждого потока есть собственные регистры и собственный стек, но другие потоки могут их использовать.
Поток — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса.

Базовая многопоточность. Процесс vs поток.

Характеристика	Процесс	Поток
Определение	Экземпляр программы в выполнении	Наименьшая единица выполнения в процессе
Изоляция	Изолированы друг от друга	Разделяют адресное пространство процесса
Ресурсы	Имеет собственные системные ресурсы	Разделяют системные ресурсы процесса
Создание	Относительно дорогая операция	Значительно дешевле, чем создание процесса
Контекст переключения	Более затратно по времени и ресурсам	Менее затратно по времени и ресурсам
Использование	Выполнение различных программ или изолированных задач	Параллельное выполнение частей одной программы

Базовая многопоточность

Многопоточность (multithreading) – свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно»

Базовая многопоточность.

Достоинства использования потоков:

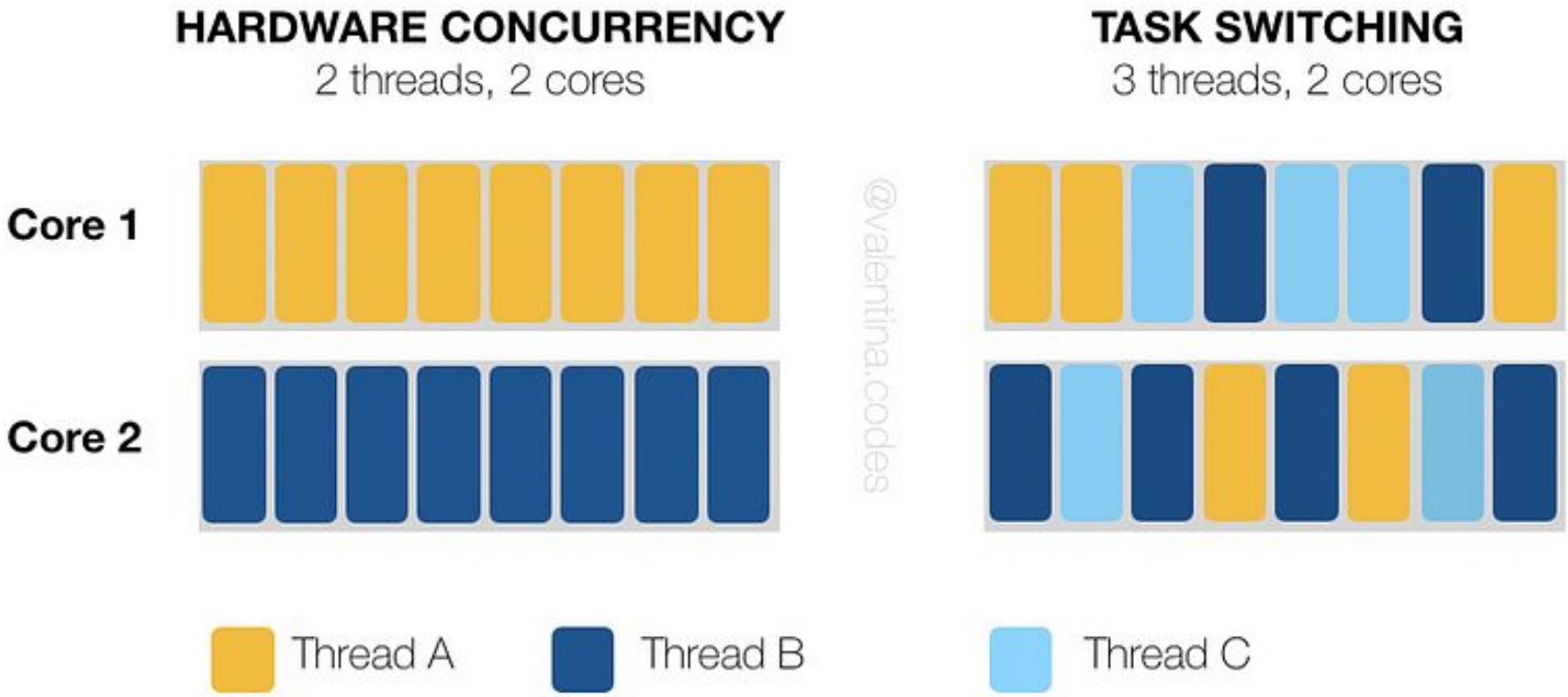
- Улучшение производительности
- Улучшение отзывчивости
- Эффективное использование ресурсов
- Повышенная масштабируемость
- Упрощение модели программирования

Базовая многопоточность

Поток может находиться в одном из трёх состояний:

- ➔ выполняемый (executing) – поток, который выполняется в текущий момент на процессоре;
- ➔ готовый (runnable) – поток ждет получения кванта времени и готов выполнять назначенные ему инструкции. Планировщик выбирает следующий поток для выполнения только из готовых потоков;
- ➔ ожидающий (waiting) – работа потока заблокирована в ожидании блокирующей операции.

Базовая многопоточность



Базовая многопоточность

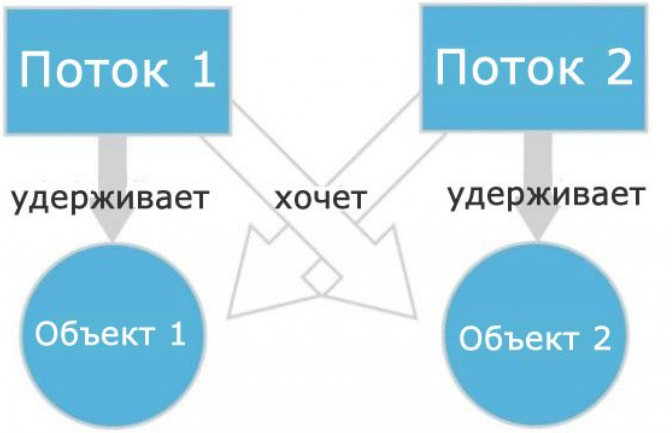
std::thread t(threadFunction); - запуск потока, в потоке выполняется функция threadFunction

t.join(); – ожидание завершения потока

t.detach(); – без ожидания завершения потока

Недостатки многопоточности

1. Состояние гонки (Race Condition)
2. Взаимоблокировка (Deadlock)



3. Сложность отладки и тестирования

Сихронизация потоков

1. Мьютексы (Mutexes)
2. Условные переменные (Condition Variables)
3. Атомарные операции (Atomics)
4. Фьючерсы и промисы (Futures and Promises)

std::mutex

Мьютекс (англ. *mutex*, от ***mut****ual* ***ex****clusion* — «взаимное исключение») — это базовый механизм синхронизации. Он предназначен для организации взаимоисключающего доступа к общим данным для нескольких потоков с использованием барьеров памяти (для простоты можно считать мьютекс дверью, ведущей к общим данным).

std::mutex

C++ STD
MUTEX

unlock() *not* guaranteed

(RAW) **MUTEX**

- `lock()`
- `unlock()`
- `try_lock()`

extensions:

TIMED_MUTEX

- `try_lock_for()`
- `try_lock_until()`

SHARED_MUTEX

Allows multiple ownerships of the mutex

RECURSIVE_MUTEX

Allows recursive locks

- `lock(), lock(), unlock(), unlock()`

unlock() guaranteed

LOCK_GUARD

Automatic mutex unlock at destruction

extensions:

UNIQUE_LOCK

- Allows multiple lock-unlock operations
- Locks a shared_mutex in exclusive mode

SHARED_LOCK

Locks a shared_mutex in shared mode

SCOPED_LOCK

Allows to lock multiple mutexes at the same time

Условные переменные (Condition Variables)

Класс `condition_variable` – это примитив синхронизации, который может использоваться для блокировки потока или нескольких потоков до тех пор, пока другой поток не изменит общую переменную (не выполнит условие) и не уведомит об этом `condition_variable`.

Условные переменные (Condition Variables)

Основные методы:

```
std::condition_variable cv;
```

```
cv.wait(lck, []{ return ready; });
```

```
cv.notify_one()
```

```
cv.notify_all();
```

std::atomic

std::atomic - это шаблонный класс из стандартной библиотеки C++, предназначенный для обеспечения безопасного доступа к разделяемым данным из нескольких потоков. Он обеспечивает атомарные операции чтения и записи, что означает, что операции с `std::atomic` гарантированно выполняются либо полностью, либо не выполняются вовсе, без промежуточных состояний. Это позволяет избежать состояний гонки и гарантирует корректное выполнение многопоточных программ.

имеет ряд методов для выполнения атомарных операций:

- load()
- store()
- exchange()
- fetch_add()
- fetch_sub()
- и другие

std::future

std::future - это часть стандартной библиотеки C++, предназначенная для работы с асинхронными операциями и возвращением результатов из асинхронных задач. Он представляет собой объект, который представляет значение, которое будет доступно в будущем, после завершения асинхронной операции