Блокчейн и умные контракты

Ионин Василий

1 Блокчейн

Блокчейн (*block chain* — цепь из блоков) — это технология, которая основана на записи некоторой информации в цепочку блоков, каждый из которых соединен с предыдущими посредством криптографии (например, через хеш-цепочки).

Обычно, каждый блок содержит криптографический хеш (в сети Bitcoin используется двойное применение алгоритма SHA256, подробнее про алгоритм можно посмотреть тут), причем хеш каждого блока в цепи зависит от хеша предыдущего блока. Цепочка считается валидной, если согласованы хеши всех блоков в цепочке (то есть хеш каждого блока такой, что если пересчитать его, учитывая хеш предыдущего блока, получится то же самое). Таким образом, достигается стойкость системы к попыткам изменения (если подменить блок в середине, все последующие хеши инвалидируются).

Чтобы обеспечить когерентность сети, у каждого пользователя блокчейна есть своя копия цепи, и участники ими постоянно обмениваются. То есть блокчейн обычно представляет из себя децентрализованную систему. Если ввести на цепочках функцию оценки, то пользователи сети смогут оставлять у себя локально цепочку с наибольшей оценкой. В качестве оценки зачастую берется количество вычислений, которые должны были быть потрачены на составление (вычисление хешей блоков) данной цепи (PoW, proof-of-work). То есть блокчейн устанавливает соглашение о том, какие события считаются достоверными.

2 Умные контракты

Умные контракты (*smart contract*) — технология, которая основана на блокчейне. Умные контракты представляют собой компьютерный алгоритм, предназначенный для формирования, контроля и предоставления информации о владении чем-либо. Зачастую умный контракт содержит в себе информацию о юридическом контракте (предмет договора, права и обязанности сторон, а также условия вступления договора в силу и условия прекращения действия договора).

Блокчейн позволяет закрепить юридический договор в системе (строится блокчейн, в котором каждый отдельный блок предсавляет собой информацию о контракте). После этого любой пользователь системы блокчейна

сможет самостоятельно посмотреть информацию о договоре и, например, проверить, выполнены ли условия вступления договора в силу. К тому же, если условия договора формализуются в виде логической формулы в программе, то стороны вынуждены сформулировать эти условия максимально точно, что может быть полезно, дабы потенциально избежать ложных интерпретаций. Если договор выражает собой право кого-то на владение ресурсом, этот ресурс может быть передан во временное владение самим блоком автоматически в момент вступления договора в силу.

Опишем примерную схему реализации умных контрактов. Мы будем писать блокчейн, который умеет исполнять программный код на каком-либо заданном языке. После того, как сторона составит контракт в виде программы на этом языке, она может добавить его в качестве блока в блокчейн, заплатив какую-то цену. Распределенная сеть потом может выполнять инструкции, прописанные в договоре, получая награду (в качестве примера сети исполнения умного контракта можно привести Ethereum). Таким образом, сеть блокчейна обеспечивает выполнение, записанных в него контрактов.

Технология умных контрактов сейчас набирает популярность. Первой страной, законодательно закрепившей смарт-контракты, стала Белоруссия.

3 Реализация блокчейна

Отмечу некоторые интересные моменты в моей реализации.

3.1 Струтура кода

Изначально я разбил файлы с исходным кодом на заголовочные, в которых я определил структуры, и файлы единиц трансляции, в которых находятся реализации.

- Файлы `block.h`/`block.cpp` отвечают за структуру элементарного блока в блокчейне;
- файлы `blockchain.h`/`blockchain.cpp` отвечают за саму структуру блокчейна;
- в `json.h` находится header-only библиотека для работы с JSON;
- в `utils.h` определена функция `hash_combine`;
- в `main.cpp` находятся тесты и точка входа в программу.

3.2 Хеширование

Чтобы составить хеш-функцию, я отдельно считаю хеш-функции от временной метки, сообщения и nonce-значения через std::hash, а затем комбинирую их с хеш-значением предыдущемго блока с помощью функции combine_hash из библиотеки Boost.

```
hash_ = std::hash<std::string>{}(message);
hash_combine(hash_, std::hash<std::time_t>{}(timestamp_));
hash_combine(hash_, std::hash<std::uint64_t>{}(nonce));
hash_combine(hash_, block.hash());
```

3.3 Сериализация и десериализация

Я сериализирую блокчейн в JSON документ с помощью header-only библиотеки nlohman::json. Хранение данных в JSON списке позволяет удобно работать с отдельными блоками.

```
nlohmann::json Block::serialize() const {
    return nlohmann::json{
            {"message",   message_},
            {"nonce",     nonce_},
            {"hash",      hash_},
            {"timestamp", timestamp_}
    };
}
```

3.4 Синхронизация

Среди приватных методов класса `BlockChain` есть функция `fork(index)`, которая копирует блокчейн с некоторого места и функция `mend(last_correct, other)`, которая подменяет блоки в блокчейне, начиная с некоторого места (оставляя все, вплоть до блока с индексом `last_correct`), на блоки блокчейна `other`.

3.5 Тестирование

В файле `main.cpp` находятся функции, тестирующие синхронизацию, а также проверяющие корректность работы сериализаторов и десериализаторов.