Лабораторная работа №3

**Тема**: «NodeJs EventLoop»

**Цель**: изучить основные понятия EventLoop, возможность по работе с событиями в NodeJs

**Теоретический материал**:

**EventLoop**

Цикл событий — это то, что позволяет Node.js выполнять неблокирующие операции ввода-вывода — несмотря на то, что JavaScript является однопоточным - путем выгрузки операций в ядро системы, когда это возможно.

Поскольку большинство современных ядер являются многопоточными, они могут обрабатывать несколько операций, выполняемых в фоновом режиме. Когда одна из этих операций завершена, ядро сообщает Node.js так что соответствующий обратный вызов может быть добавлен в очередь опроса для последующего выполнения.

Когда Node.js запускается, он инициализирует цикл событий, обрабатывает предоставленный сценарий ввода (или переходит в REPL), который может выполнять асинхронные вызовы API, планировать таймеры или вызывать process.nextTick(), затем начинает обработку цикла событий.

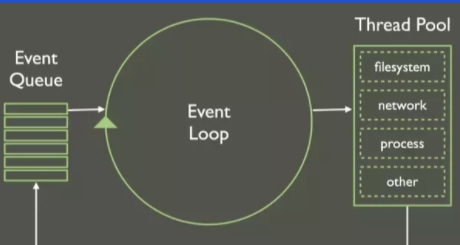


Рисунок 1 – схема работы EventLoop

Это простейшее представление, как выглядит Event Loop. Есть определённая очередь событий, есть бесконечный цикл событий, который синхронно выполняет операции из очереди, и он распределяет их дальше.

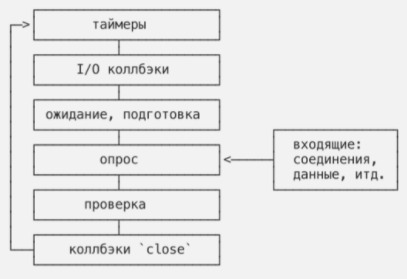


Рисунок 2 – Очередь событий

*1 фаза (Таймеры)* - Данная фаза выполняется непосредственно Event Loop'ом здесь просто обновляется время, когда начал работать Event Loop.

*2 фаза (I/O-callback'и)* - Когда демультиплексор событий выполняет чтение какого-либо файла и ставит выполнение callback'а в очередь, это как раз соответствует этапу I/O-callback. Здесь выполняются callback'и для неблокирующего ввода/вывода, т. е. это именно те функции, которые используются после запроса в базу данных или другой ресурс или на чтение/запись файла. Они выполняются именно на данной фазе.

*3 фаза (ожидание, подготовка)* — внутренние операции для callback'ов, по сути, мы не можем влиять на фазу, только косвенно. Есть process.nextTick, его callback может ненамеренно быть исполнен на фазе «ожидание, подготовка». process.nextTick выполняется на текущей фазе, т. е., по сути, process.nextTick может сработать абсолютно на любой фазе. Какого-то готового инструмента, чтобы запустить код на фазе «ожидание, подготовка», в Node.js нет.

*4 фаза (опрос)* - Здесь выполняется весь наш код, который мы пишем на JS. Первоначально все запросы, которые мы делаем, попадают именно сюда, и именно здесь Node.js может быть заблокирована. Если сюда попадёт какая-либо тяжёлая операция по вычислению, то на этом этапе наше приложение может просто зависнуть и ожидать, пока не выполнится данная операция.

*5 фаза (проверка)* - В Node.js есть таймер setImmediate, его callback'и выполняются на этой фазе.

*6 фаза (последняя)* - Например, web-сокету нужно закрыть соединение, на этой фазе будет вызван callback этого события

**Практическая часть**:

Задание 1 Написать скрипт, который наглядно будет демонстрировать работу каждой из фаз EventLoop:

1.1 Создать три таймера SetTimeOut с разным временем срабатывания. При срабатывании таймера выводить время, прошедшее с момента запуска таймера.

1.2 Применить операции ввода-вывода для записи и чтения файла. После окончания операции ввода-вывода выводить время, прошедшее с момента начала операции.

1.3 Добавить вызов SetImmediate. Вывести время, прошедшее с момента вызова.

Задание 2 Зарегистрировать в системе пользовательское событие. Сгенерировать и обработать пользовательское событие с выводом информации о событии.

Задание 3 Промоделировать ситуацию, в которой Вы являетесь продавцом товара (описание товара в лабораторной №2). На обновление списка товара подписать нескольких клиентов, которым будет приходить сообщение при добавлении нового товара или модификации старого. Клиенты должны логгировать информацию от продавца в своих журналах в виде файла с занесением времени.

**Ссылки для детального просмотра**:

1. EventLoop - <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/>
2. Event - [https://nodejs.org/api/events.html#event-newlistener/](https://nodejs.org/api/events.html" \l "event-newlistener)