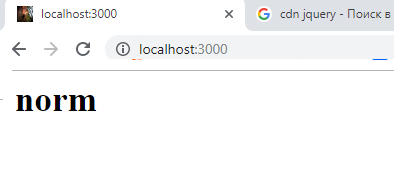
Лабораторная работа 03а

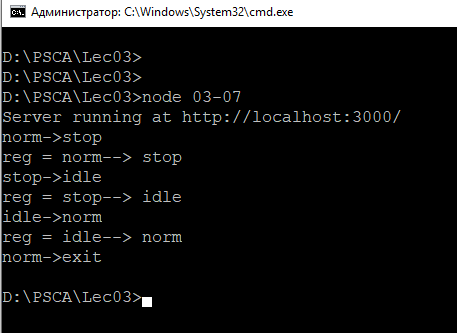
ПвИ

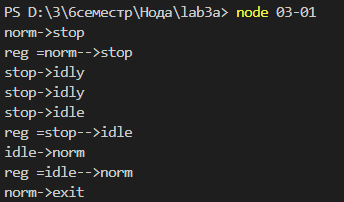
**Задание 01**

1. Разработайте серверное приложение **03-01**, которое на запрос [**http://localhost:5000**](http://localhost:5000)возвращает страницу, отражающую состояние приложения (см. рис.).



1. Приложение может находиться в четырех состояниях: **norm**, **stop**, **test**, **idle**.
2. Состояние приложение переключается с помощью стандартного системного ввода, который назначен на консоль. Консоль в приглашении (prompt) указывает текущее состояние приложения.
3. Пользователь может ввести новое состояние (**norm**, **stop**, **test**, **idle**). При корректном вводе состояния осуществляется переключение состояния приложения.
4. При ошибочном вводе режима ошибочная введенная последовательность символов просто отображается, но переключение режима не осуществляется.
5. Допускается ввод состояния exit, которое приводит к завершению приложения.
6. См. рис.





**Задание 02**

1. Разработайте серверное приложение **03-02**, которое на GET-запрос вида [**http://localhost:5000**](http://localhost:5000)**/fact?k=3** возвращает ответ, в теле которого содержится сообщение в json-формате вида {k:3, fact:6}, где **k** – полученное в качестве параметра значение, а **fact** – значение факториала.
2. Для расчета факториала используйте рекурсивный алгоритм.
3. Проверьте работоспособность приложения с помощью **POSTMAN**.

Метод url.parse() принимает строку URL-адреса, анализирует ее и возвращает объект URL-адреса с каждой частью адреса в качестве свойств.

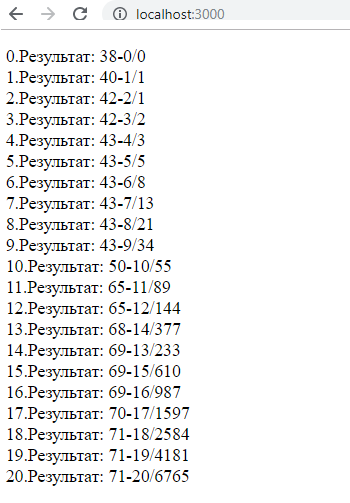
Функция **parseInt()** принимает строку в качестве аргумента и возвращает целое число в соответствии с указанным основанием системы счисления.

**Метод Number.isInteger () определяет, является ли переданное значение целым числом**.

**Задание 03**

1. Доработайте приложение **03-02** таким образом, чтобы на GET-запрос приложение отправляло HTML-страницу, содержимое которой формировалось бы с помощью JS.
2. JS в цикле x = 1,…,20 с помощью функции fetch делает GET-запросы к [**http://localhost:5000/fact?k=x**](http://localhost:5000/fact?k=x) и содержимое ответа выводит в окно браузера, примерно так, как это представлено на следующем рисунке.

[localhost:5000](http://localhost:5000/)



1. Результаты вычислений должны иметь следующий вид **t-k/fact**,где t – количество миллисекунд, прошедшее с момента начала работы цикла запросов, k – параметр, пересылаемый серверу, **fact** факториал k.
2. Запустите приложение и запишите общую продолжительность всего цикла запросов.

[localhost:5000](http://localhost:5000/)

**Задание 04**

1. Разработайте приложение **03-04** на основе приложения **03-02,** но функцию для вычисления факториала реализуйте асинхронной с помощью механизма **process.nextTick**.
2. Выполните аналогичные заданию 3 замеры.

**Тиком (tick)** называют один полный проход цикла событий. Передавая функцию методу **process.nextTick(),** мы сообщаем системе о том, что эту функцию нужно вызвать после завершения текущей итерации цикла событий, до начала следующей.

 Этот метод стоит использовать тогда, когда нужно обеспечить выполнение некоего кода в самом начале следующей итерации цикла событий.

**Задание 05**

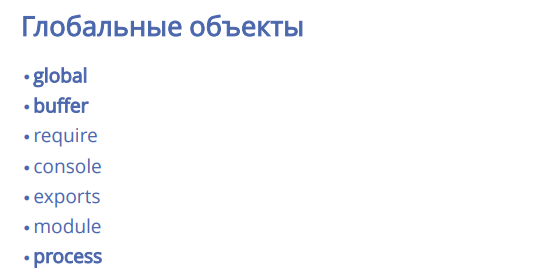
1. Разработайте приложение **03-05** на основе приложения **03-02,** но функцию для вычисления факториала реализуйте асинхронной с помощью механизма **setImmediate**.
2. Выполните аналогичные заданию 3 замеры.

**Задание 06.** Ответьте на следующие вопросы.

1. Перечислите основные глобальные объекты Node.js и поясните их предназначение.
2. Поясните понятие «асинхронная функция».
3. Поясните понятие стандартные «системные потоки».
4. Поясните назначение функций **process.nextTick**, **setImmediate**. Поясните в чем их разница.

**Задание 06.** Ответьте на следующие вопросы.

1. **Перечислите основные глобальные объекты Node.js и поясните их предназначение.**



**Global** - специальный объект, который предоставляет доступ к глобальным, то есть доступным из каждого модуля приложения, переменным и функциям.

**Buffer** – глобальный объект, предназначенный для работы с двоичными данными: набором октетов.

**Process** - глобальный объект, который предоставляет информацию о текущем процессе Node.js и контролирует его.

**Require** – используют для загрузки модуля, обычно присваивая результат его работы какой-то переменной.

**Console** – используется для вывода сообщений различных уровней в stdout и stderr. Используется синхронно, когда информация предназначена для вывода в файл или на терминал, и асинхронно, когда данные передаются в пайп.

**Exports** – ключевое слово exports ссылается на глобальный объект, доступный в каждом модуле Node.js. Все функции и объекты, хранящиеся в объекте exports модуля, становятся открытыми, когда другие модули Node.js импортируют этот объект.

**Module** – он используется для включения определенного модуля с использованием метода require().

1. **Поясните понятие «асинхронная функция».**

Асинхронность представляет возможность одновременно выполнять сразу несколько задач. Выполнение на функции не блокируется, а идет дальше.

**Асинхронная функция** - особый вид функции или метода, которые можно приостановить на полпути выполнения. Это отличается от обычных синхронных функций и методов, которые либо выполняются до завершения, либо вызывают ошибку, либо никогда не возвращаются.

1. **Поясните понятие стандартные «системные потоки».**

**Поток представляет собой последовательность данных, поступающих во времени от источника и направляющихся к месту назначения.**

Как только поток открывается, данные передаются блоками (**chunks**) из своего источника в процесс, потребляющий их. Поступая из файла, каждый символ или байт считывается по одному; поступая с клавиатуры, каждое нажатие клавиши передает данные потоком.

Входные потоки также называются **читаемыми потоками**, что означает, что они предназначены для чтения данных из источника.

Данные представляют собой последовательность элементов, поступающих во времени (например, символы или байты).

**читаемый поток - process.stdin**

**записываемый поток - process.stdout**

мы можем передавать любой контент из любого читаемого источника в любое записываемое место назначения.

Каждому потоку нужна **функция \_read.** Эта функция вызывается, когда поток проверяет данные. Это правильное место, чтобы запустить другие механизмы и вставить обновленное содержимое в поток.

1. **Поясните назначение функций process.nextTick, setImmediate. Поясните в чем их разница.**

**Функция обратного вызова**, переданная **setImmediate(),** будет выполнена на следующей итерации цикла событий.

**Функци**я, переданная **process.nextTick()** выполнится после завершения текущей итерации цикла событий. То есть, такая функция всегда будет выполняться до функции, выполнение которой запланировано с помощью setTimeout() или setImmediate().