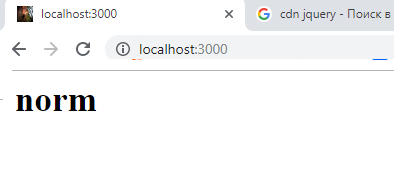
Лабораторная работа 03а

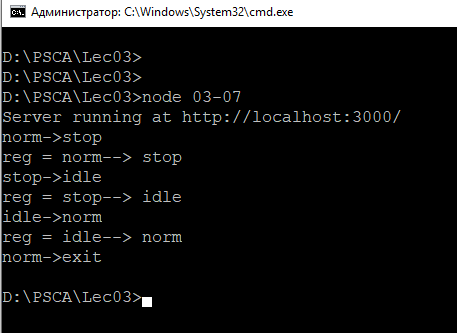
СТРWП

**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 приложения 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 ошибочно введенная последовательность символов просто отображается, но переключение режима не осуществляется.
5. Допускается ввод состояния exit, которое приводит к завершению приложения (см. рис.)

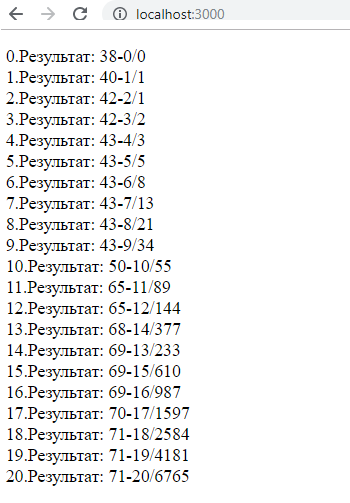


**Задание 02**

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**.

**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. В цикле x = 1,…,20 с помощью функции fetch сделайте GET-запросы к [**http://localhost:5000/fact?k=x**](http://localhost:5000/fact?k=x) и содержимое ответа выведите в окно браузера, примерно так, как это представлено на следующем рисунке.



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.

**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.

**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 назначение.
2. Поясните понятие «асинхронная функция».
3. Поясните понятие «стандартные системные потоки».
4. Поясните назначение функций **process.nextTick**, **setImmediate**. Поясните в чем их разница.

Основные глобальные объекты Node.js включают в себя:

\_\_dirname: строковое свойство, содержащее абсолютный путь к директории, в которой находится исполняемый файл.

\_\_filename: строковое свойство, содержащее абсолютный путь к исполняемому файлу.

Buffer: класс, представляющий двоичные данные.

console: объект, предоставляющий методы для вывода сообщений в консоль.

exports: объект, используемый для экспорта функций и переменных из модуля.

module: объект, представляющий текущий модуль.

process: объект, представляющий текущий процесс.

require(): функция, используемая для загрузки модулей.

Асинхронная функция - это функция, которая выполняется асинхронно и возвращает Promise. Она может содержать ключевое слово await, которое приостанавливает выполнение функции до тех пор, пока не будет выполнен Promise.

Стандартные системные потоки - это потоки ввода/вывода, которые связывают программу с ее окружением. В Node.js есть три стандартных потока: stdin, stdout и stderr. stdin используется для ввода данных в программу, stdout - для вывода данных из программы, а stderr - для вывода сообщений об ошибках.

Функция process.nextTick() используется для планирования выполнения функции после завершения текущей операции I/O. Функция setImmediate() используется для планирования выполнения функции после завершения текущего цикла событий. Основное отличие между ними заключается в том, что функция process.nextTick() выполняется перед любыми другими операциями I/O, а функция setImmediate() выполняется после завершения текущего цикла событий. setImmediate - макрозадача, макрозадачи выполняются по одной за один проход цикла FIFO Callback, process.nextTick - микрозадача, на каждом проходе выполняются все накопившиеся микрозадачи.