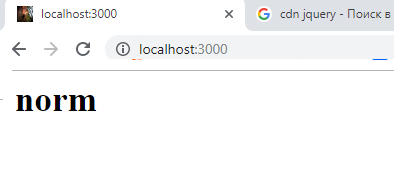
Лабораторная работа 03

ПСКП

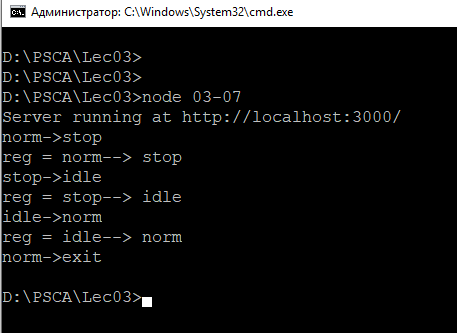
ПОИТ-3

**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.



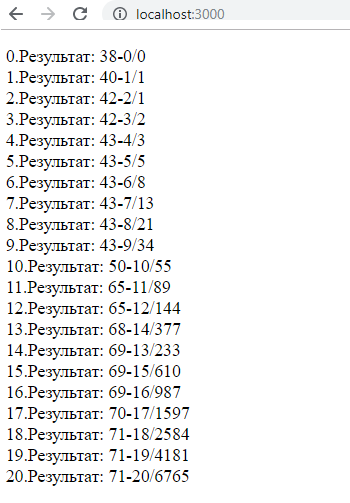
ПОИТ-3

**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. 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.



1. Результаты вычислений должны иметь следующий вид **t-k/fac**,где t – количество миллисекунд прошедшее с момента начала работы цикла запросов, k – параметр пересылаемый серверу, **fac** факториал k.
2. Выполните приложение запишите общую продолжительность всего цикла запросов.
3. Запустите приложение поочередно еще в двух вкладках браузера и запишите продолжительность всего цикла запросов для каждой вкладки.
4. Запустите приложение одновременно в трех вкладках браузера и запишите продолжительность всего цикла запросов для каждой вкладки.

**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.

## **process.nextTick()**

Метод process.nextTick() по-особому взаимодействует с циклом событий. 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. Использование данного метода выглядит так:

process.nextTick(() => {

 //выполнить какие-то действия

})

Предположим, цикл событий занят выполнением кода текущей функции. Когда эта операция завершается, JavaScript-движок выполнит все функции, переданные process.nextTick() в ходе выполнения предыдущей операции. Используя этот механизм, мы стремимся к тому, чтобы некая функция была бы выполнена асинхронно (после текущей функции), но как можно скорее, без постановки её в очередь.  
  
Например, если воспользоваться конструкцией setTimeout(() => {}, 0) функция будет выполнена на следующей итерации цикла событий, то есть — гораздо позже, чем при использовании в такой же ситуации process.nextTick(). 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.

## **setImmediate()**

Ещё одной функцией, предоставляемой Node.js для асинхронного выполнения кода, является setImmediate(). Вот как ей пользоваться:

setImmediate(() => {

 //выполнить некий код

})

Функция обратного вызова, переданная setImmediate(), будет выполнена на следующей итерации цикла событий.  
  
Чем setImmediate() отличается от setTimeout(() => {}, 0) (то есть, от таймера, который должен сработать как можно скорее) и от process.nextTick()?  
  
Функция, 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().  
  
Вызов функции setTimeout() с установленной задержкой в 0 мс очень похож на вызов setImmediate(). Порядок выполнения функций, переданных им, зависит от различных факторов, но и в том и в другом случаях коллбэки будут вызваны на следующей итерации цикла событий.

**Задание 06.Ответьте на следующие вопросы**

1. Перечислите основные свойства глобальных объектов 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 разница.