Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

По курсу: "Архитектура ЭВМ"

Студент Наместник Анастасия
Группа ИУ7-53Б
Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема Взаимодействие серверов. Дочерние процессы. Аргументы командной строки

Студент:	_____	Наместник А.А.
	подпись, дата	Фамилия, И.О.
Преподаватель:	_____	Попов А. Ю.
	подпись, дата	Фамилия, И. О.

Москва — 2020 г.

TASK_7.

Цель работы:

— Изучать и реализовать взаимодействие между серверами.
— Изучать и реализовать дочерние процессы.
— Изучать и реализовать process.argv.
— Изучить prolog.

Задание 1

Создать сервер А. На стороне сервера хранится файл с содержимым в формате JSON. При получении запроса на /insert/record идёт добавление записи в файл. При получении запроса на /select/record идёт получение записи из файла. Каждая запись хранит информацию о машине (название и стоимость).

Создать сервер Б. На стороне сервера хранится файл с содержимым в формате JSON. Каждая запись в файле хранит информацию о складе и массиве машин, находящихся на данном складе. То есть каждая запись хранит в себе название склада (строку) и массив названий машин (массив строк). При получении запроса на /insert/record идёт добавление записи в файл. При получении запроса на /select/record идёт получение записи из файла.

Создать сервер С. Сервер выдаёт пользователю страницы с формами для ввода информации. При этом сервер взаимодействует с серверами А и Б. Реализовать для пользователя функции:

создание нового типа машины получение информации о стоимости машины по её типу создание нового склада с находящимися в нём машинами получение информации о машинах на складе по названию склада Реализовать удобный для пользователя интерфейс взаимодействия с системой (использовать поля ввода и кнопки).

Листинг 1 — Код программы. TASK_7. Сервер А

```
1   "use strict";
2
3   // импорт библиотеки
4   const express = require("express");
5   const fs = require("fs");
6
7
8   // запускаем сервер
9   const app = express();
10  const port = 5003;
11  app.listen(port);
12  console.log("Server on port " + port);
13
14  // заголовки для ответа
```

```
15      app.use(function (req, res, next) {
16          res.header("Cache-Control", "no-cache, no-store, must-revalidate");
17          res.header("Access-Control-Allow-Headers", "Origin,
                X-Requested-With, Content-Type, Accept");
18          res.header("Access-Control-Allow-Origin", "*");
19          next();
20      });
21
22      // Загрузка тела.
23      function loadBody(request, callback) {
24          let body = [];
25          request.on('data', (chunk) => {
26              body.push(chunk);
27          }).on('end', () => {
28              body = Buffer.concat(body).toString();
29              callback(body);
30          });
31      }
32
33      // Приём запроса.
34      app.post("/insert/record", function (request, response) {
35          loadBody(request, function (body) {
36              // Получаем данные.
37              const obj = JSON.parse(body);
38              const type = obj.type;
39              const price = obj.price;
40
41              // Открываем файл и парсим.
42              const fileName = "cars.json";
43              const objInfo = fs.readFileSync(fileName, "utf-8");
44              const infoJson = JSON.parse(objInfo);
45              let answer = "Model exists!";
46
47              let flag = true;
48              // Ищем модель.
49              for (let i in infoJson) {
50                  if (infoJson[i].type === type) {
51                      // console.log(infoJson[i]);
52                      flag = false;
53                      break;
54                  }
55              }
56
57              // Добавляем в файл информацию,
58              // Если такой модели еще нет.
59              if (flag) {
60                  infoJson.push({ type, price })
```

```
61                  fs.writeFileSync(fileName, JSON.stringify(infoJson, null,
                        4));
62                  answer = "Model added";
63              }
64
65              response.end(JSON.stringify({ answer: answer }));
66          });
67      });
68
69      // Приём запроса.
70      app.post("/select/record", function (request, response) {
71          loadBody(request, function (body) {
72              // Получаем данные.
73              const obj = JSON.parse(body);
74              const type = obj.type;
75
76              // Открываем файл и парсим.
77              const fileName = "cars.json";
78              const objInfo = fs.readFileSync(fileName, "utf-8");
79              const infoJson = JSON.parse(objInfo);
80
81              // Ответ пользователю.
82              let answer = "Model does not exist!";
83
84              // Ищем модель.
85              for (let i in infoJson) {
86                  if (infoJson[i].type === type) {
87                      // console.log(infoJson[i]);
88                      answer = infoJson[i];
89                      break;
90                  }
91              }
92
93              response.end(JSON.stringify({ answer: JSON.stringify(answer)
                    }));
94          });
95      });
```

Листинг 2 — Код программы. TASK_7. Сервер B

```
1   "use strict";
2
3   // импорт библиотеки
4   const express = require("express");
5   const fs = require("fs");
6
7
```

```
8     // запускаем сервер
9     const app = express();
10    const port = 5002;
11    app.listen(port);
12    console.log("Server on port " + port);
13
14    // заголовки для ответа
15    app.use(function (req, res, next) {
16        res.header("Cache-Control", "no-cache, no-store, must-revalidate");
17        res.header("Access-Control-Allow-Headers", "Origin,
              X-Requested-With, Content-Type, Accept");
18        res.header("Access-Control-Allow-Origin", "*");
19        next();
20    });
21
22    // Загрузка тела.
23    function loadBody(request, callback) {
24        let body = [];
25        request.on('data', (chunk) => {
26            body.push(chunk);
27        }).on('end', () => {
28            body = Buffer.concat(body).toString();
29            callback(body);
30        });
31    }
32
33    // Приём запроса.
34    app.post("/insert/record", function (request, response) {
35        loadBody(request, function (body) {
36            // Получаем данные.
37            const obj = JSON.parse(body);
38            const stock_name = obj.stock_name;
39            const car_array_str = obj.car_array;
40
41            // Открываем файл и парсим.
42            const fileName = "storage.json";
43            const objInfo = fs.readFileSync(fileName, "utf-8");
44            const infoJson = JSON.parse(objInfo);
45            let answer = "Storage exists!";
46
47            let flag = true;
48            // Ищем склад.
49            for (let i in infoJson) {
50                if (infoJson[i].stock_name === stock_name) {
51                    // console.log(infoJson[i]);
52                    flag = false;
53                    break;
```

```
54                  }
55              }
56
57              // Добавляем в файл информацию,
58              // Если такой модели еще нет.
59              if (flag) {
60                  let car_array = car_array_str.split(" ");
61                  infoJson.push({ stock_name, car_array })
62                  fs.writeFileSync(fileName, JSON.stringify(infoJson, null,
                        4));
63                  answer = "Model added";
64              }
65
66              response.end(JSON.stringify({ answer: answer }));
67          });
68      });
69
70      // Приём запроса.
71      app.post("/select/record", function (request, response) {
72          loadBody(request, function (body) {
73              // Получаем данные.
74              const obj = JSON.parse(body);
75              const name_stock_find = obj.name_stock_find;
76
77              // Открываем файл и парсим.
78              const fileName = "storage.json";
79              const objInfo = fs.readFileSync(fileName, "utf-8");
80              const infoJson = JSON.parse(objInfo);
81
82              // Ответ пользователю.
83              let answer = "Model does not exist!";
84
85              // Ищем модель.
86              for (let i in infoJson) {
87                  if (infoJson[i].stock_name === name_stock_find) {
88                      // console.log(infoJson[i]);
89                      answer = infoJson[i];
90                      break;
91                  }
92              }
93
94              response.end(JSON.stringify({ answer: JSON.stringify(answer)
                    }));
95          });
96      });
```

Листинг 3 — Код программы. TASK_7.Сервер C

```
"use strict";

// импорт библиотек
const express = require("express");
const request = require("request");
const fs = require("fs");

const ENCODING = "utf-8"

// запускаем сервер
const app = express();
const port = 5000;
app.listen(port);
console.log(`Server on port ${port}`);

// Отправка статических файлов.
const way = __dirname + "/static";
app.use(express.static(way));

// заголовки в ответ клиенту
app.use(function (req, res, next) {
    res.header("Cache-Control", "no-cache, no-store, must-revalidate");
    res.header("Access-Control-Allow-Headers", "Origin,
       X-Requested-With, Content-Type, Accept");
    res.header("Access-Control-Allow-Origin", "*");
    next();
});

// функция для отправки POST запроса на другой сервер
function sendPost(url, body, callback) {
    // задаём заголовки
    const headers = {};
    headers["Cache-Control"] = "no-cache, no-store, must-revalidate";
    headers["Connection"] = "close";
    // отправляем запрос
    request.post({
        url: url,
        body: body,
        headers: headers,
    }, function (error, response, body) {
        if (error) {
            callback(null);
        } else {
            callback(body);
        }
```

```
45          });
46      }
47
48      app.get("/", (_request, response) => {
49          const fileContent = fs.readFileSync("static/" + "index.html",
                ENCODING);
50          response.end(fileContent);
51      });
52
53      // принимаем GET запрос и отправляем POST запрос на другой сервер
54      app.get("/set_info_car/", (request, response) => {
55          const type = request.query.field_type_car;
56          const price = request.query.field_price_car;
57
58          //Формируем запрос на сервер A
59          sendPost("http://localhost:5003/insert/record", JSON.stringify(
60              { type, price }
61          ), function (answerString) {
62              const answerObject = JSON.parse(answerString);
63              const answer = answerObject.answer;
64              response.end("Answer: " + answer);
65
66              // Либо ту же страницу возвращать.
67              //  const fileContent = fs.readFileSync("static/" +
                    "index.html", ENCODING);
68              // response.end(fileContent);
69          });
70      });
71
72      app.get("/get_info_car/", (request, response) => {
73          const type = request.query.field_type_car_find;
74
75          sendPost("http://localhost:5003/select/record", JSON.stringify(
76              { type }
77          ), function (answerString) {
78              const answerObject = JSON.parse(answerString);
79              const answer = answerObject.answer;
80              response.end("Answer: " + answer);
81          });
82      });
83
84      // принимаем GET запрос и отправляем POST запрос на другой сервер
85      app.get("/set_info_stock/", (request, response) => {
86          const stock_name = request.query.field_stock_name;
87          const car_array = request.query.field_car_array;
88
89          sendPost("http://localhost:5002/insert/record", JSON.stringify(
```

```
90              { stock_name, car_array }
91          ), function (answerString) {
92              const answerObject = JSON.parse(answerString);
93              const answer = answerObject.answer;
94              response.end("Answer: " + answer);
95          });
96      });
97
98      app.get("/get_info_stock/", (request, response) => {
99          const name_stock_find = request.query.field_name_stock_find;
100
101         sendPost("http://localhost:5002/select/record", JSON.stringify(
102             { name_stock_find }
103         ), function (answerString) {
104             const answerObject = JSON.parse(answerString);
105             const answer = answerObject.answer;
106             response.end("Answer: " + answer);
107         });
108     });
```

Задание 2

Написать скрипт, который принимает на вход число и считает его факториал. Скрипт должен получать параметр через process.argv.

Написать скрипт, который принимает на вход массив чисел и выводит на экран факториал каждого числа из массива. Скрипт принимает параметры через process.argv.

При решении задачи вызывать скрипт вычисления факториала через execSync.

Листинг 4 — Код программы. TASK_7. Задание 2. Главный процесс

```
1  "use strict";
2
3  // Запуск:
4  // npm start 5 6 7 8 9
5
6  //В Node js дочерние процессы создаются для выполнения ресурсоемких операци
       й,
7  //которые во время выполнения блокируют цикл событий основного процесса.
8  //Создаваемые дочерние процессы полностью независимы от родительского проце
       сса
9  //и имеют свои собственные экземпляры V8 и выделенные мощности процессора и
       объем памяти.
10
11 // импортируем библиотеку для работы с процессами
12 const execSync = require('child_process').execSync;
```

```
13
14 // Начиная со второго аргумента
15 // Идут наши параметры (ранее пути идут)
16 const MY_ARG = 2
17 const OPTIONS = { encoding: 'utf8' };
18
19 // Функция, для считывания аргументов
20 // Переданных в командной строке.
21 function readArgv(array) {
22     let i = MY_ARG;
23
24     while (process.argv[i])
25         array.push(parseInt(process.argv[i++]));
26
27     return array;
28 }
29
30 // Функция, вызывающая дочерний процесс
31 // Для каждого элемента из массива array.
32 // Дочерний процесс в свою очередь
33 // Считает факториал числа.
34 function arrayFactorial(array) {
35     let cmd;
36
37     for (let i in array) {
38         cmd = `node factorial ${array[i]}`
39         console.log(execSync(cmd, OPTIONS))
40     }
41 }
42
43
44 function main() {
45     let array = [];
46     readArgv(array);
47     arrayFactorial(array);
48 }
49
50 main();
```

Листинг 5 — Код программы. TASK_2. Задание 2. Дочерний процесс.

```
1     "use strict";
2
3     // Функция, которая вычисляет факториал
4     // Числа, переданного аргументом командной строки.
5     function factorial() {
6         let num = parseInt(process.argv[2]);
```

```
        let result = 1;

        for (let i = 1; i <= num; i++)
            result *= i;

        console.log(result);
    }

    factorial();
```

TASK_8

С клавиатуры считываются числа A и B. Необходимо вывести на экран все числа Фибоначчи, которые принадлежат отрезку от A до B.

Листинг 6 — Числа Фибоначчи

```
writeNumber(X, A) :- X_NEW is X,
    ((X_NEW >= A) -> write(X_NEW), write(" "); write("")).
fib(N1, N2, A, B) :-
N1 > 0,
F is N1 + N2,
writeNumber(N1, A),
N2 =< B,
fib(N2, F, A, B).

f :- read(A), read(B), fib(1,1,A,B).
```

С клавиатуры считываются числа A и B. Необходимо вывести на экран все числа, квадратный корень которых является целым числом. При этом, необходимо вывести только числа, которые принадлежат отрезку от A до B.

квадратный корень которых является целым числом

Листинг 7 — Числа

```
ok.
pow(F1,F2,F3,A,B):- F3 > B.
pow(F1,F2,F3,A,B):-
    F3 < A,
    F4 is F1 * F2,
    F5 = F1 + 1,
    pow(F5,F5,F4,A,B).
pow(F1,F2,F3,A,B):- write(F3), write("\n"), F4 is F1 * F2, F5 = F1 + 1,
   pow(F5,F5,F4,A,B).
f :-read(A), read(B),pow(0,0,0,A,B),ok.

f :- read(A), read(B), fib(1,1,A,B).
```

Вывод:

— Мы изучили и реализовали взаимодействие между серверами.
— Изучили и реализовали дочерние процессы.
— Изучили и реализовали process.argv.
— Изучили prolog.

Пример работы:

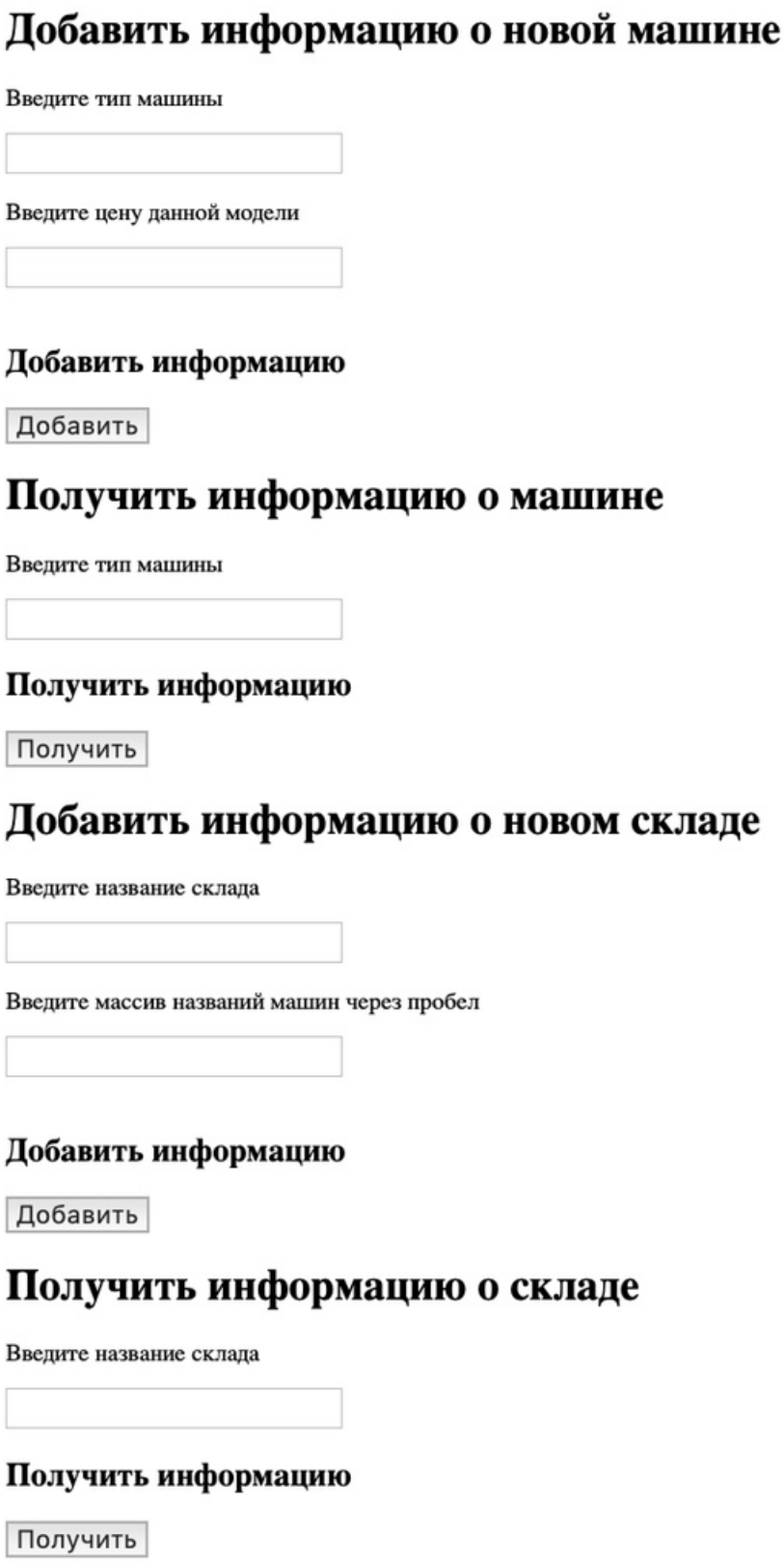

Рисунок 0.1 — Пример работы программы

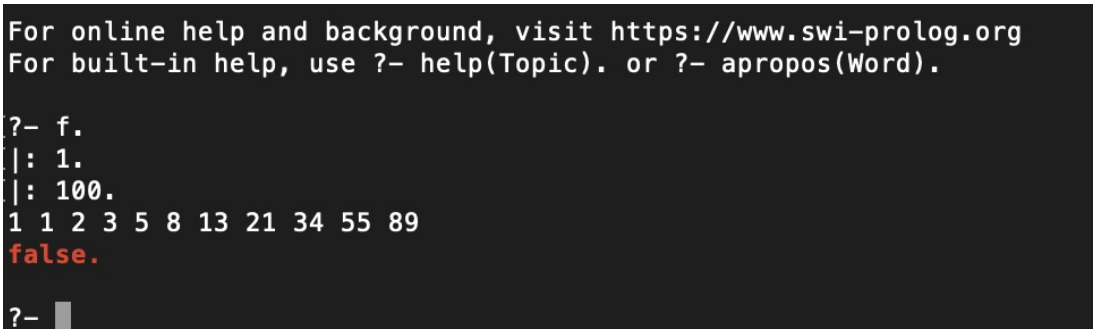

Рисунок 0.2 — Пример работы программы

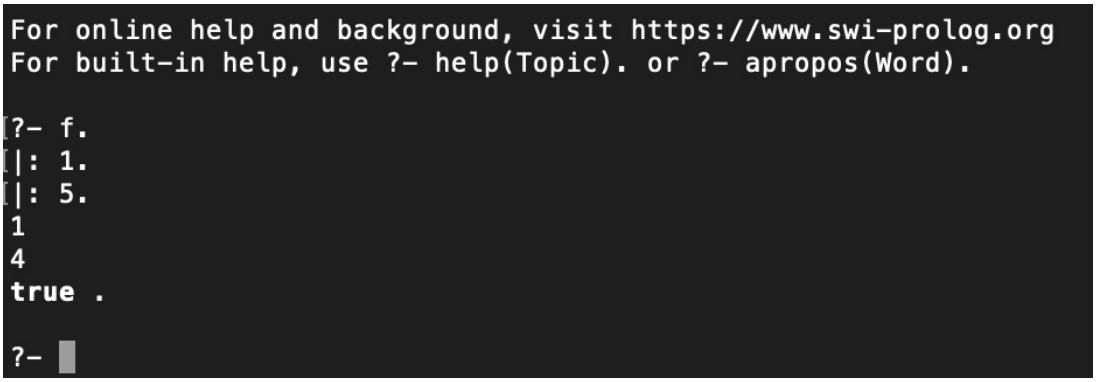

Рисунок 0.3 — Пример работы программы