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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение эвм и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Разработка статического сервера»

Студент **ИУ7-71Б** _____ **Волков Г.В.**

Руководитель _____

2023 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
« » 2023 г.

ЗАДАНИЕ
на выполнение курсовой работы

по теме

«Разработка статического сервера»

Студент группы **ИУ7-71Б**

Волков Георгий Валерьевич

Направленность КР

учебная

Источник тематики

Курсовая работа кафедры

График выполнения НИР: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание
Разработать статический веб-сервер. В качестве мультиплексора использовать poll. Сервер должен реализовывать многопоточную обработку запросов с использованием пула потоков.

Оформление научно-исследовательской работы:
Расчетно-пояснительная записка на **12-20** листах формата А4.

Дата выдачи задания « » 2023 г.

Руководитель курсовой работы _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ **Волков Г.В.**
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Веб–сервер — сервер, принимающий и обрабатывающий HTTP–запросы, обычно от браузеров, и отдающий HTTP–ответы, как правило, с HTML–страницей или медиафайлами.

Целью работы — написать статический сервер для отдачи контента с диска.

Для достижения поставленной в работе цели предстоит решить следующие задачи:

— формализовать требования к серверу;

— исследовать предметную область веб серверов;

— спроектировать программное обеспечение;

— реализовать программное обеспечение;

— провести сравнительный анализ, написанного программного обеспечения, с nginx.

1 Аналитические раздел

Данный раздел включает в себя формализацию задачи и анализ протокола HTTP, паттерна thread pool и мультиплексора poll.

1.1 Требования к серверу

Для реализуемого сервера выдвигается следующий требования:

— поддержка запросов GET и HEAD и статусов 200, 403, 404;

— ответ на неподдерживаемые запросы статусом 405;

— выставление content type в зависимости от типа файла;

— корректная передача файлов размером в 100мб;

— защита от выхода за пределы root директории сервера;

— стабильная работа сервера.

1.2 Протокол HTTP

HTTP (протокол передачи гипертекста) — это протокол прикладного уровня. Реализуется в двух частях приложений: клиентской и серверной. Клиенты и сервер общаются друг с другом, обмениваясь сообщениями. Протокол определяет структуру этих сообщений и порядок обмена [1].

HTTP использует TCP в качестве базового транспортного протокола. Сначала HTTP–клиент инициирует TCP–соединение с сервером, по нему клиент отправляет HTTP–запрос. HTTP–сервер принимает запрос и отправляет HTTP–ответ. Также за счёт использования TCP в HTTP гарантируется доставка всех сообщений [1].

Сервер отправляет запрошенные файлы без сохранения какой-либо информации о нем. Если некоторый объект будет многократно запрошен, то он будет каждый раз полностью отправлен в ответе. Поэтому HTTP называют протоколом без сохранения состояния [1].

HTTP работает с непостоянными и постоянными соединениями. В первом случае каждая пара запрос-ответ отправляются через отдельные соединения, а во втором через одно. Непостоянные соединения применяются по умолчанию в версии 1.0 HTTP, в то время как постоянные соединения в версии HTTP 1.1. Краткосрочные соединения имеют два больших недостатка: требуется значительное время на установку нового соединения, и то, что эффективность TCP-соединения улучшается только по прошествии некоторого времени от начала его использования. У постоянных соединений есть свои недочёты; даже работая вхолостую, они потребляют ресурсы сервера, а при высокой нагрузке могут проводиться DoS-атаки. Соединения управляются заголовком Connection. Значение close указывает, что клиент или сервер хотели бы закрыть соединение. keep-alive указывает, что клиент хотел бы сохранить соединение активным [2].

Постоянные соединения поддерживаю конвейерную обработку. Это процесс отсылки последовательных запросов по одному постоянному соединению не дожидаясь ответа. Таким образом избегают задержки соединения. Не все типы запросов HTTP позволяют конвейерную обработку: только идемпотентные методы, а именно GET, HEAD, PUT и DELETE, можно перезапускать безопасно: в случае сбоя содержимое конвейерной передачи можно просто повторить. Демонстрация работы HTTP с различными типами соединений представлена на рисунке 1.1 [2].

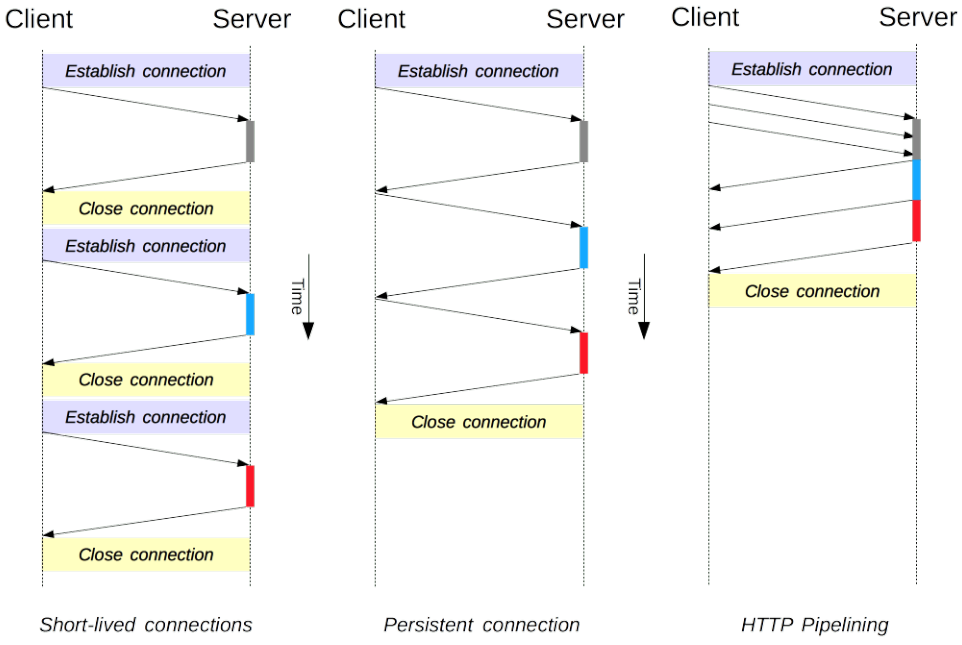

Рисунок 1.1 – HTTP с различными типами соединений

HTTP–запрос представлен в обычном текстовом формате ASCII. Первая строка HTTP-сообщения называется строкой запроса; следующие строки называются строками заголовка. В строке запроса содержатся: метод, URL, версия протокола. Заголовки записываются как пары имя: значение и разделены символом новой строки. Формат запроса представлен на рисунке 1.2 [1].

Рисунок 1.2 – Общий формат сообщения–запроса HTTP

HTTP–ответ тоже представляется в виде текста, воспринимаемого человеком и состоит из строки состояния, заголовков и тела. Строка состояния содержит версию протокола, код состояния и соответствующее сообщение. Формат запроса представлен на рисунке 1.3 [1].

Рисунок 1.3 – Общий формат сообщения-ответа протокола HTTP

1.3 Паттерн thread pool

Пул потоков — то шаблон проектирования программного обеспечения, обеспечивающий параллельное выполнение программ. Он поддерживает множество потоков, ожидающих поступления задач для одновременного выполнения. Количество потоков настраиваемый параметр, как правило, оптимальные количество совпадает с количеством логических ядер. Малое количество потоков может работать недостаточно быстро, а чрезмерно большое количество приводит к пустой трате вычислительных ресурсов [3].

Преимуществ пула потоков по сравнению с созданием нового потока для каждой задачи является то, что затраты на создание и уничтожение потоков ограничиваются первоначальным созданием, а не при выполнении каждой задачи. Но это требует дополнительной синхронизации потоков [3].

Для выполнения задания необходимо поставить его в очередь, читаемую воркерами. Результат, если он требуется, забирается из второй очереди, которая заполняется воркерами. Сами же потоки крутятся в бесконечном цикле, на каждой итерации которого проверяют наличие работ на выполнения и собственно выполняют их. Каждое обращение к очереди выполняется в режиме монопольного доступа для избежания состояния гонки, что достилается за счёт использования примитивов синхронизацию. Схема работы представлена на рисунке 1.4 [3].

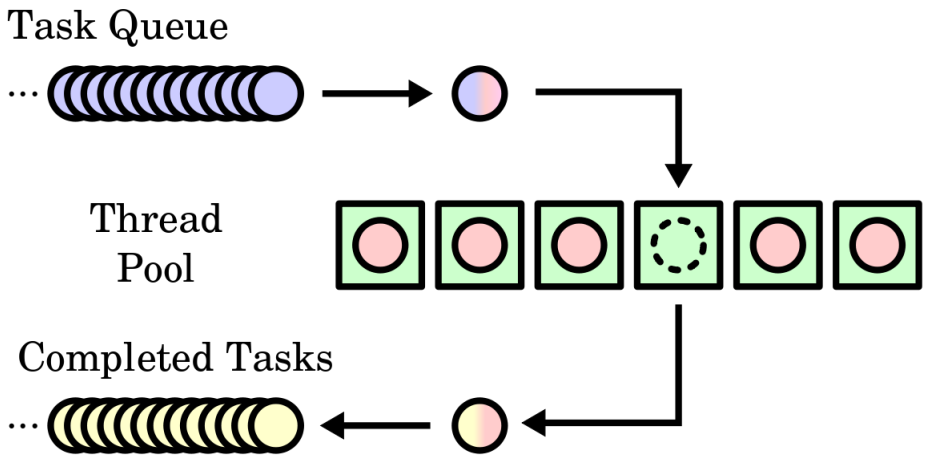

Рисунок 1.4 – Схема работы thread pool

1.4 Мультиплексирование

Сокет — это программный интерфейс для обмена данными данными между процессорами как на одном компьютере, так и на различных. Представляют собой конечную точку соединения. При сетевом взаимодействии сокеты являются абстракцией над IP–адресом и портом.

Возможность сообщить ядру, о необходимости получать информации о том, что на одном или нескольких дескрипторов из множества выполнилось какое-либо условие ввода–вывода, называется мультиплексированием. poll — функция мультиплексирования, которая использует блокировку ввода-вывода с мультиплексированием, алгоритм работы представлен на рисунке. Основными достоинствами её являются неограниченное количество слушаемых сокетов и неизменность массива слушаемых сокетов, что освобождает от его постоянной перезаписи. pool принимает массив содержащий номера дескрипторов и интересующие события.

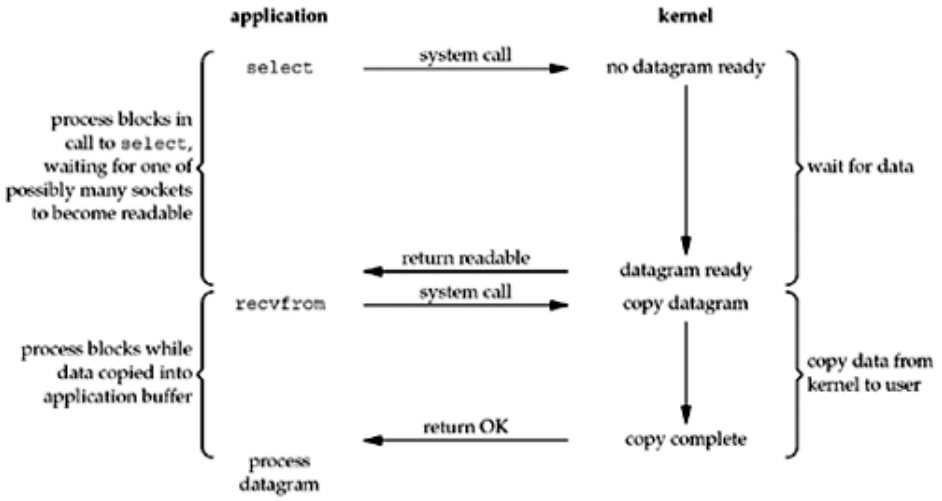

Рисунок 1.5 – Модель мультиплексирования ввода–вывода

Вывод

В данном раздел была формализована задача и рассмотрены протокола HTTP, паттерн thread pool и мультиплексор poll.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритмов обработки запросов

.

2.1 Обработка запроса

Обработкой появления какого-либо события на сокете занимается основной поток программы, который может принять новое соединение или при наличии данных на сокете поставить задачу в очередь для thread pool. Схема его работы изображена на рисунке 2.1.

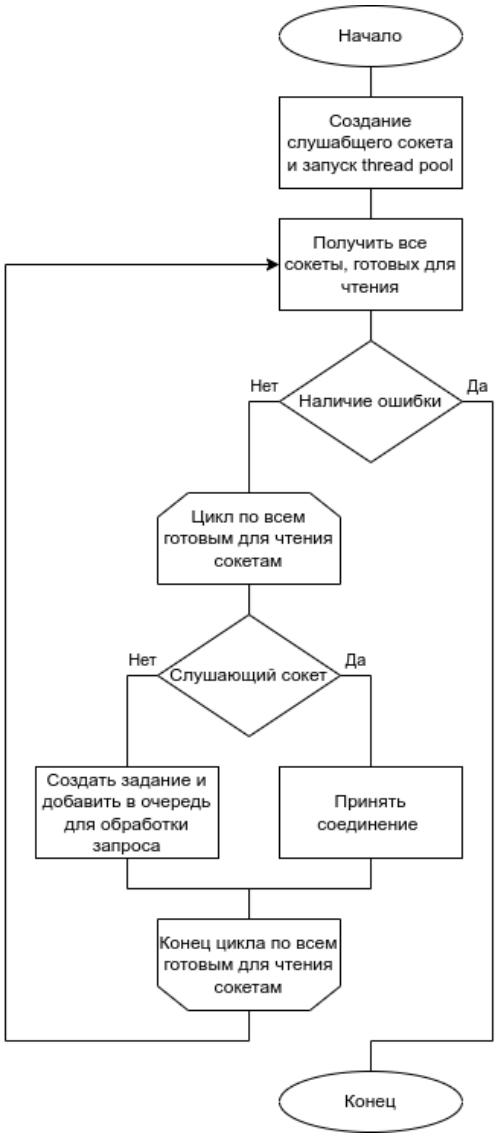

Рисунок 2.1 – Схема работы thread pool

Поток из пула просыпается при поступлении задачи в очередь и обрабатывает HTTP–запрос. Схема его работы изображена на рисунке 2.2.

Рисунок 2.2 – Схема работы thread pool

Вывод

В данном разделе разработаны и представлены схемы алгоритмов обработки запросов.

3 Конструкторский раздел

В данном разделе будут представлены технические аспекты реализации программы.

3.1 Средства реализаци

Для реализации ПО был выбран язык C [4] так как он был указан в задании и обладает всем необходимым функционалом для реализации требуемого программного обеспечения. В качестве среды разработки была выбрана Clion [5].

3.2 Реализация сервера

В листинге 3.1 представленная реализация функции запуска сервера.

Листинг 3.1 – Запуск сервера

```
1  int run_http_server_t(http_server_t *server) {
2      log_info("Server starting");
3      log_info("Host: %s", server->host);
4      log_info("Port: %d", server->port);
5      log_info("Work dir: %s", server->wd);
6
7      server->listen_sock = listen_net(server->host, server->port);
8      if (server->listen_sock < 0) return -1;
9
10     if (run_tpool_t(server->pool) != 0) return -1;
11
12     long maxi = 0, nready;
13     server->clients[0].fd = server->listen_sock;
14     server->clients[0].events = POLLIN;
15
16     while (1) {
17         nready = poll(server->clients, maxi + 1, -1);
18         if (nready < 0) {
19             log_fatal(ERR_FSTR, "poll error", strerror(errno));
```

```
20              return -1;
21          }
22
23          if (server->clients[0].revents & POLLIN) {
24              int client_sock = accept_net(server->listen_sock);
25              if (client_sock < 0) continue;
26
27              long i = 0;
28              for (i = 1; i < server->cl_num; ++i) {
29                  if (server->clients[i].fd < 0) {
30                      server->clients[i].fd = client_sock;
31                      break;
32                  }
33              }
34              if (i == server->cl_num) {
35                  log_error("too many connections");
36                  continue;
37              }
38              server->clients[i].events = POLLIN;
39
40              if (i > maxi) maxi = i;
41              if (--nready <= 0) continue;
42          }
43          for (int i = 1; i <= maxi; ++i) {
44              if (server->clients[i].fd < 0) continue;
45
46              if (server->clients[i].revents & (POLLIN | POLLERR)) {
47                  task_t *task = new_task_t(handle_connection,
                        server->clients[i].fd, server->wd);
48                  if (task == NULL) continue;
49
50                  add_task(server->pool, task);
51                  server->clients[i].fd = -1;
52
53                  if (--nready <= 0) break;
54              }
55          }
56      }
57  }
```

В листинге 3.2 представленная реализация функции воркера thread

pool, в которой он дожидается поступления задания в очередь, забирает его и запускается обработку запроса. Само задание представляет собой структуру хранящую указатель на функцию обработки запросов и указатели на параметры для неё. Она представленная в листинге 3.3.

Листинг 3.2 – Получение задачи из очереди

```
1  void *routine(void *args) {
2      routine_args_t *r_args = (routine_args_t *) args;
3      tpool_t *pool = r_args->pool;
4      int num = r_args->num;
5      free(args);
6
7      task_t task;
8      char name[15] = "";
9      sprintf(name, "thread-%d", num);
10     thread_name = name;
11
12     while (1) {
13         pthread_mutex_lock(pool->q_mutex);
14
15         s_wait(pool->sem, pool->q_mutex);
16
17         if (pool->stop == 1) {
18             pthread_mutex_unlock(pool->q_mutex);
19             log_debug("stopped");
20             break;
21         }
22
23         int rc = pop(pool->queue, &task);
24         if (rc != -1) {
25             log_debug("work taken (q len = %d)", pool->queue->len);
26         }
27         pthread_mutex_unlock(pool->q_mutex);
28         if (rc < 0) continue;
29
30         task.handler(task.conn, task.wd);
31         log_info("routine for task finished");
32     }
33
34     pthread_exit(NULL);
35 }
```

Листинг 3.3 – Получение задачи из очереди

```
typedef struct routine_args_t {
    tpool_t *pool;
    int num;
} routine_args_t;
```

В листинге 3.4 представленная реализация функции обработки HTTP запроса.

Листинг 3.4 – Обработка запроса

```
void handle_connection(int clientfd, char *wd) {
    request_t req;
    char *buff = calloc(REQ_SIZE, sizeof(char));
    if (buff == NULL) {
        log_error(ERR_FSTR, "failed alloc req buf",
            strerror(errno));
        return;
    }

    log_debug("handle_connection started");
    if (read_req(buff, clientfd) < 0) {
        send_err(clientfd, INT_SERVER_ERR_STR);
        close(clientfd);
        free(buff);
        return;
    }
    log_debug("read_req");

    if (parse_req(&req, buff) < 0) {
        send_err(clientfd, BAD_REQUEST_STR);
        close(clientfd);
        free(buff);
        return;
    }
    if (req.method == BAD) {
        log_error("unsupported http method");
        send_err(clientfd, M_NOT_ALLOWED_STR);
```

```
27            close(clientfd);
28            free(buff);
29            return;
30        }
31
32        process_req(clientfd, &req, wd);
33
34        close(clientfd);
35        log_debug("handle_connection finished");
36
37        free(buff);
38    }
```

4 Исследовательский раздел

В данном разделе представлены технические характеристики, демонстрация работы программы и её сравнение с nginx.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

— Операционная система Manjaro Linux 86_64 Xfce 4.18 [6].

— Оперативная память: 8 Гбайт.

— Процессор: 11th Gen Intel i5-1135G7 (8) @ 4.200 Гц [7].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

4.3 Исследование времени обработки запроса

На рисунке 4.1 представлено сравнение реализованного сервера с nginx. Замеры проводились с помощью программы ApacheBench. Из результатов видно, что написанная программа обрабатывает запросы быстрее nginx в среднем на 35%.

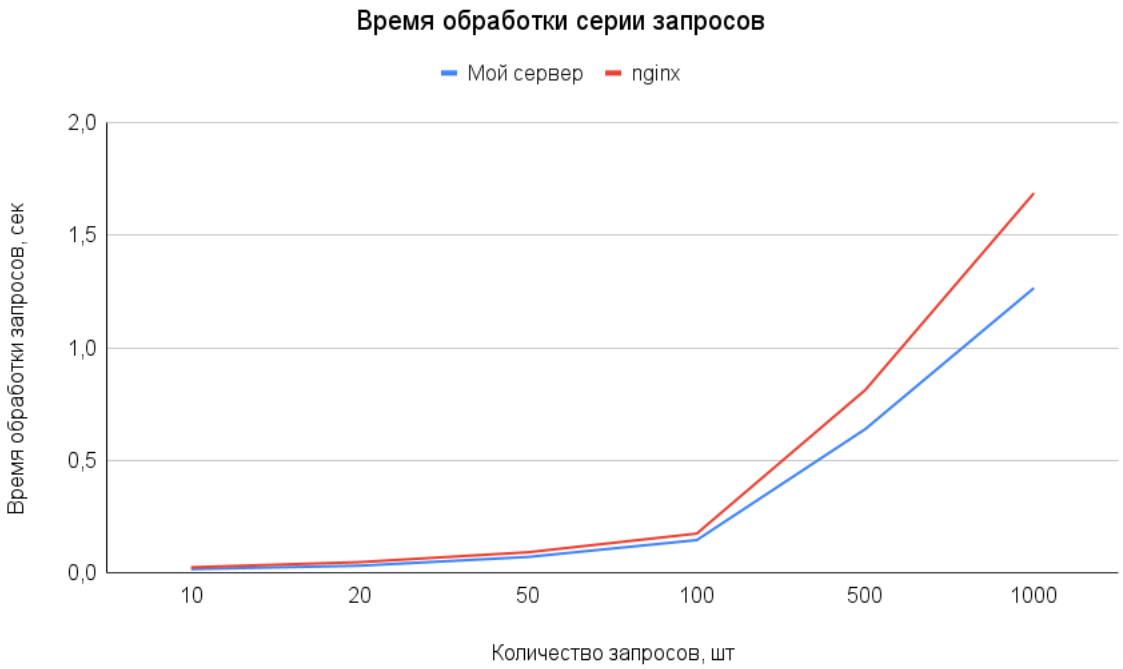

Рисунок 4.1 – Сравнение с nginx

4.4 Вывод

Были представлены технические характеристики, демонстрация работы программы и её сравнение с nginx, которое показало, что реализованный сервер работает быстрее в среднем на 35%.

ЗАКЛЮЧЕНИЕ

Цель, которая была поставлена в начале курсовой работы, была достигнута: написан статический сервер для отдачи контента с диск.

В ходе выполнения курсовой работы были решены все задачи:

— формализованы требования к серверу;

— исследована предметную область веб серверов;

— спроектировано программное обеспечение;

— реализовано программное обеспечение;

— проведён сравнительный анализ, написанного программного обеспечения, с nginx;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Джеймс Куроуз, Кит Росс Компьютерные сети: Нисходящий подход. — 6-е изд. — М.: Издательсво «Э», 2016. — 912 С.

2. Connection management in HTTP/1.x [Электронный ресурс]. — URL: `https://developer.mozilla.org/ru/docs/Web/HTTP/Connection_management_in_HTTP_1.x` (дата обращения: 12.12.2023).

3. Энтони Уильямс Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. — М.: ДМК Пресс, 2012. — 672 С.

4. The GNU C Reference Manual [Электронный ресурс]. — URL: `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html` (дата обращения: 12.12.2023).

5. CLion Кросс-платформенная IDE для C и C++ [Электронный ресурс]. — URL: `https://www.jetbrains.com/ru-ru/clion/` (дата обращения: 12.12.2023).

6. Manjaro Linux [Электронный ресурс]. — URL: `https://manjaro.org/` (дата обращения: 12.12.2023).

7. Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. — URL: `https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html` (дата обращения: 12.12.2023).