



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»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

*«Разработка сервера для отдачи статического
содержимого с диска»*

Студент ИУ7И-72Б
(Группа)

(Подпись, дата)

Фам Минь Хиеу
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Клочков М. Н.
(И. О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка 22 с., 10 рис., 0 табл., 4 источн., 1 прил.
ВЕБ-СЕРВЕР, THREAD-POOL, PSELECT, NGINX, С.

Цель работы — разработка статического сервера раздачи файлов с диска. Для параллельной обработки использовать thread pool. В качестве мультиплексора использовать pselect.

В результате работы была проанализирована архитектура пула потоков, спроектирована схема алгоритма работы веб-сервера. Разработан статический веб-сервер. Проведено сравнение разработанного сервера с nginx.

СОДЕРЖАНИЕ

РЕФЕРАТ	3
ВВЕДЕНИЕ	5
1 Аналитический раздел	6
1.1 Архитектура thread-pool	6
1.2 Системный вызов pselect	8
2 Конструкторский раздел	9
2.1 Схема алгоритма работы сервера	9
3 Технологический раздел	10
3.1 Средство реализации	10
3.2 Реализация сервера	10
3.3 Демонстрация работы программы	15
4 Исследовательский раздел	18
4.1 Технические характеристики	18
4.2 Результаты исследования	19
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ А	22

ВВЕДЕНИЕ

Сервер — это компьютер, подключенный к компьютерной сети или Интернету, имеющий статический IP-адрес и обладающий высокой вычислительной мощностью. На нем устанавливаются программное обеспечение, которое позволяет другим компьютерам получать доступ к услугам и ресурсам и запрашивать их. Сервер используется для хранения и обработки данных в компьютерной сети или в интернет-среде. Сервер является основой всех услуг в Интернете. Любая услуга в Интернете, такая как веб-сайт, приложение, игра должна работать через сервер.

Целью данной работы является разработка статического сервера для отдачи файлов с диска. Для многопоточной обработки с использованием пула потоков (thread pool) [1]. Для реализации мультиплексирования использовать системный вызов `pselect` [2].

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) проанализировать архитектуру «пул потоков» и системный вызов `pselect`;
- 2) спроектировать алгоритм работы сервера;
- 3) разработать сервер раздачи статических данных;
- 4) сравнить разрабатываемый сервер с `nginx`.

1 Аналитический раздел

1.1 Архитектура thread-pool

При изучении написания многопоточных приложений, начинающие разработчики часто используют следующий подход для обработки параллельных задач:

- 1) создание нового потока;
- 2) передача параметров и запуск потока для выполнения задачи
- 3) завершение потока и получение результата.

Одной из наиболее распространенных задач для начинающих является разработка чат-приложения. Процесс реализации аналогичен описанному выше:

- 1) открытие серверного сокета и ожидание подключений на этом сокете;
- 2) когда новое клиентское соединение устанавливается, создается поток, который ожидает данные на этом соединении. Если данные поступают, они обрабатываются, или отправляются данные на другую сторону через сокет;
- 3) после завершения работы обеих сторон соединение разрывается, и поток завершается.

При большом количестве подключений клиентов создается множество потоков, что приводит к значительным затратам времени на переключение контекста, то есть времени, необходимого для перехода от одного потока к другому. Чем больше времени уходит на переключение, тем меньше времени остается на обработку задач, что приводит к снижению производительности системы.

При создании слишком большого количества потоков система может исчерпать доступные ресурсы ЦП, необходимых для обработки задач. Если время подключения короткое, это означает, что потоки создаются, выполняются и завершаются быстро. Процесс создания и завершения потока является

время затратным, поэтому при коротком времени использования это также приводит к неэффективности.

Если время подключения длительное, но в процессе этого подключения клиент в основном находится в состоянии ожидания, то есть не выполняет никаких действий, это означает, что система вынуждена поддерживать поток неэффективно.

Таким образом, для решения вышеупомянутых недостатков в большинстве случаев используется модель пула потоков (thread pool) [1].

Архитектура пула потоков представлена на рисунке 1.1.

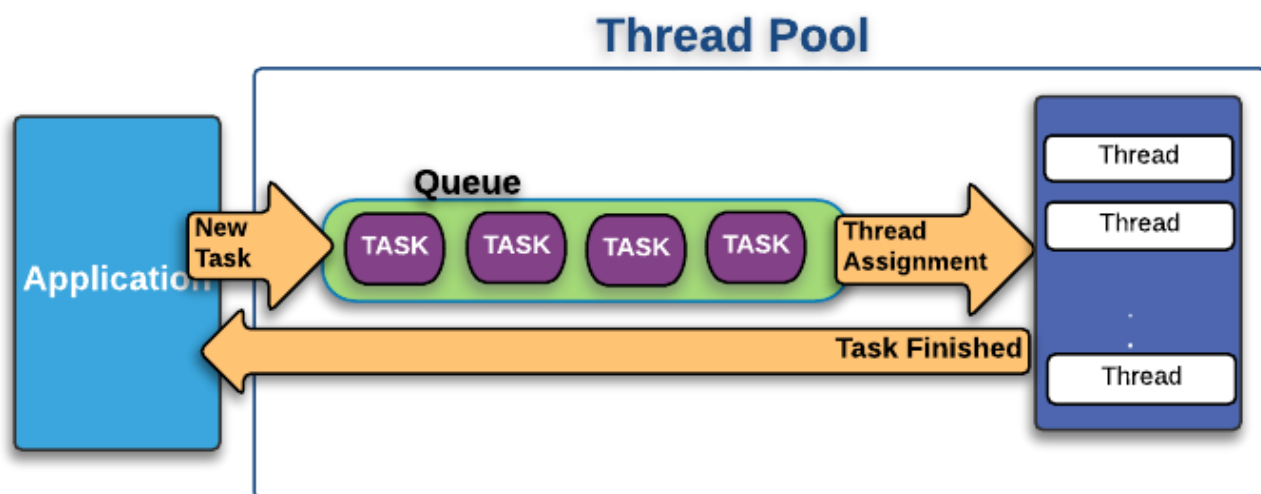


Рисунок 1.1 – Архитектура пула потоков

Модель пула потоков имеет следующие преимущества:

- 1) контроль количества потоков: Можно управлять числом потоков в пуле, причем это количество может быть изменено во время работы приложения;
- 2) снижение затрат на создание и завершение потоков: Процесс инициализации и завершения потоков минимизируется, что позволяет экономить ресурсы и время;

Некоторые замечания при использовании пула потоков:

- 1) очередь задач должна быть потокобезопасной, то есть должна обеспечивать доступ из нескольких потоков без возникновения ошибок;
- 2) каждая задача должна содержать достаточную информацию для обработки, а данные в задаче и потоке должны быть независимыми.

1.2 Системный вызов pselect

Функция `pselect()` предназначена для мониторинга активности в наборе сокетов и/или идентификаторов очередей сообщений до тех пор, пока не истечет время ожидания [2]. Они позволяют определить, имеются ли условия для чтения, записи или обработки исключений для любых из этих сокетов и очередей сообщений. Данные вызовы также работают с обычными файловыми дескрипторами, каналами и терминалами.

Модель неблокирующего синхронного ввода-вывода с применением системного вызова `pselect()`, представлена на рисунке 1.2.

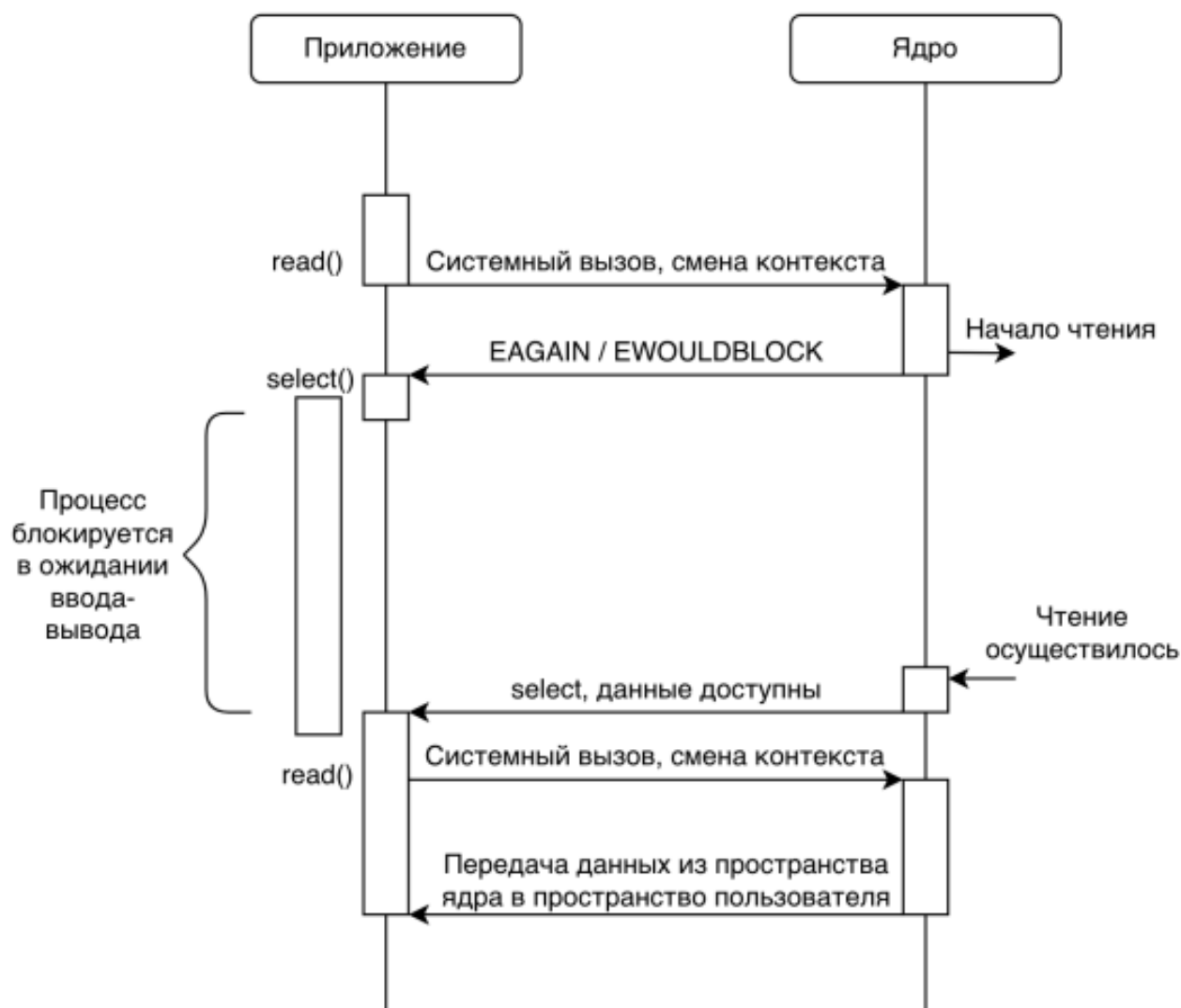


Рисунок 1.2 – Модель неблокирующего синхронного ввода-вывода

2 Конструкторский раздел

2.1 Схема алгоритма работы сервера

На рисунке 2.1 представлена схема алгоритма работы сервера. Разница во времени обработки становится более заметной при большом количестве запросов. Например, при 1000 запросах разработанный сервер показывает результат в 2.13 раз быстрее, в то время как при 200 запросов эта разница 1.17.

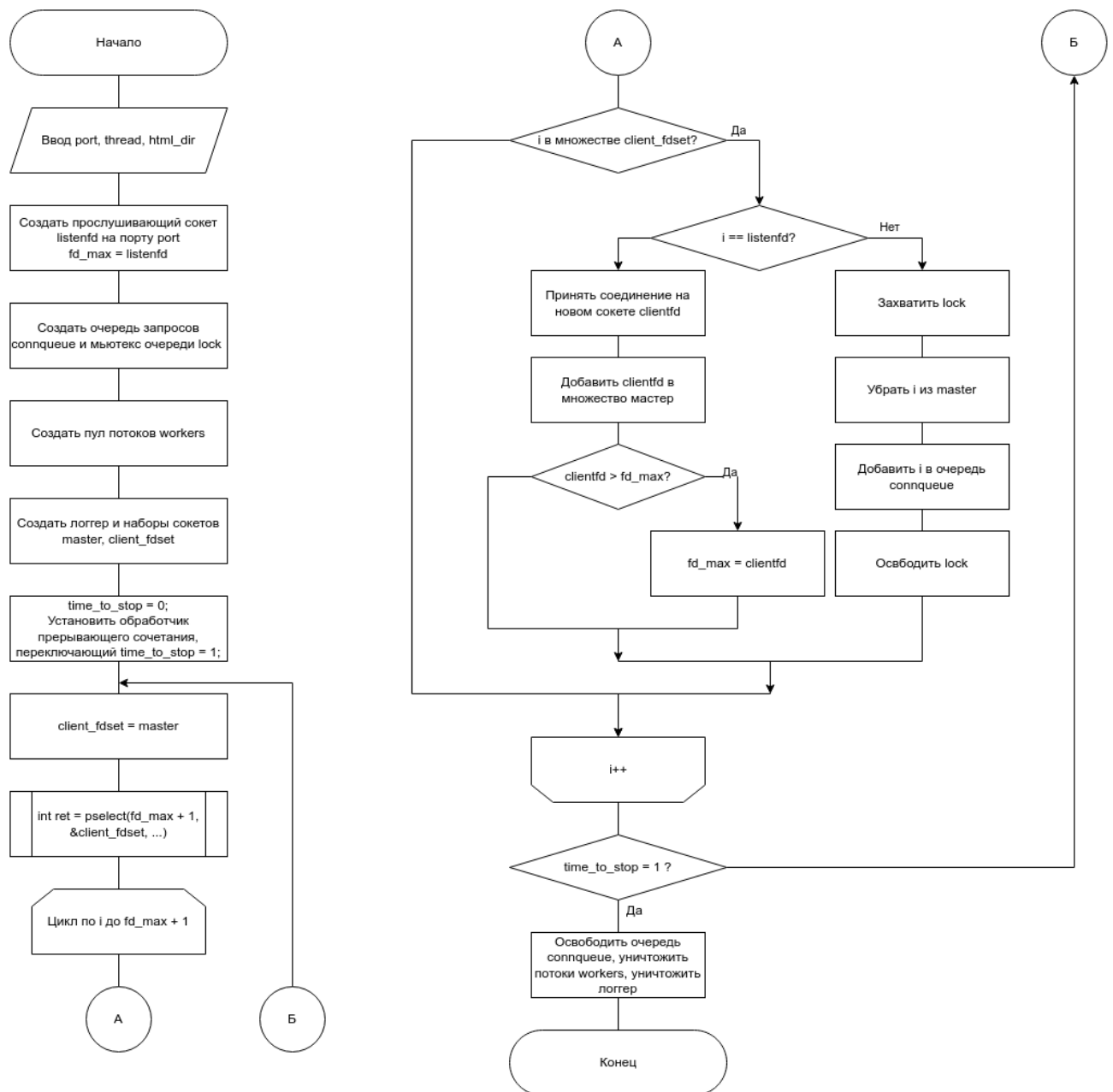


Рисунок 2.1 – Схема алгоритма работы сервера

3 Технологический раздел

3.1 Средство реализации

Согласно требованиям к работе в качестве языка программирования для реализации веб-сервера был выбран язык C. В качестве среды для разработки была выбрана среда Visual Studio Code.

3.2 Реализация сервера

Реализация логики работы сервера представлена в листинге ??.

Листинг 3.1 – Реализация логики работы сервера

```
void *thread_funcion(void *arg) {
    int n = *((int *) arg);
    httpRequestResponsePtr request;
    fd_set read_fds, write_fds;
    struct timeval timeout;
    while (1) {
        // printf("in thread hehehe\n");
        int max_fd;
        FD_ZERO(&read_fds);
        FD_ZERO(&write_fds);
        max_fd = -1;

        for (int i = 0; i < N; i++) {
            if (fdArray[n][i] != -1) {
                FD_SET(fdArray[n][i], &read_fds);
                FD_SET(fdArray[n][i], &write_fds);
                // printf("%d was setted\n", i + 1);
                if (fdArray[n][i] > max_fd) {
                    max_fd = fdArray[n][i];
                }
            }
        }

        // printf("maxfd= %d\n", max_fd);
        timeout.tv_sec = 3;
        timeout.tv_usec = 0;
        int activity = pselect(max_fd + 1, &read_fds,
                               &write_fds, NULL, &timeout, NULL);
        // printf("actv= %d\n", activity);
    }
}
```

```

if (activity < 0) {
    logMessage(LOG_ERROR, strerror(errno), 0);
    continue;
}
if (activity == 0)
{
    printf("time out. no event\n");
}

for (int i = 0; i < maxConnections; i++) {
    // printf("socket %d\n", httpRRArray[i].socket);
    if (httpRRArray[i].socket != -1) {
        if (FD_ISSET(httpRRArray[i].socket, &read_fds)) {
            printf("read fds\n");
            request = &httpRRArray[i];
            switch(request->state)
            {
                case STATE_CONNECT:
                    request->readBuffer->bytesRead = 0;
                    request->readBuffer->bytesToRead =
                        MAXPAYLOAD;
                    readRequest(request);
                    break;
                case STATE_READ:
                    readRequest(request);
                    break;
            }
        }
        if (httpRRArray[i].state == STATE_SEND &&
            (httpRRArray[i].socket, &write_fds)) {
            printf("write fds\n");

            sendResponse(&httpRRArray[i]);
        }
        if (httpRRArray[i].state == STATE_COMPLETE ||
            httpRRArray[i].state == STATE_ERROR) {
            shutdown(httpRRArray[i].socket, SHUT_RDWR);
            close(httpRRArray[i].socket);
            for (int k = 0; k < N; k++) {
                if (fdArray[n][k] ==
                    httpRRArray[i].socket)

```

```

                                fdArray[n][k] = -1;
                                }
                                httpRRArray[i].socket = -1;
                                }
                                }
                                }
                                }
                                }
}

```

Реализация цикла обработки запросов представлена в листинге 3.2.

Листинг 3.2 – Реализация цикла обработки запросов

```

void *thread_funcion(void *arg) {
    int n = *((int *) arg);
    httpRequestResponsePtr request;
    fd_set read_fds, write_fds;
    struct timeval timeout;
    while (1) {
        // printf("in thread hehehe\n");
        int max_fd;
        FD_ZERO(&read_fds);
        FD_ZERO(&write_fds);
        max_fd = -1;

        for (int i = 0; i < N; i++) {
            if (fdArray[n][i] != -1) {
                FD_SET(fdArray[n][i], &read_fds);
                FD_SET(fdArray[n][i], &write_fds);
                // printf("%d was setted\n", i + 1);
                if (fdArray[n][i] > max_fd) {
                    max_fd = fdArray[n][i];
                }
            }
        }

        // printf("maxfd= %d\n", max_fd);
        timeout.tv_sec = 3;
        timeout.tv_usec = 0;
        int activity = pselect(max_fd + 1, &read_fds,
                               &write_fds, NULL, &timeout, NULL);
        // printf("actv= %d\n", activity);

        if (activity < 0) {

```

```

        logMessage(LOG_ERROR, strerror(errno), 0);
        continue;
    }
    if (activity == 0)
    {
        printf("time out. no event\n");
    }

    for (int i = 0; i < maxConnections; i++) {
        // printf("socket %d\n", httpRRArray[i].socket);
        if (httpRRArray[i].socket != -1) {
            if (FD_ISSET(httpRRArray[i].socket, &read_fds)) {
                printf("read fds\n");
                request = &httpRRArray[i];
                switch(request->state)
                {
                    case STATE_CONNECT:
                        request->readBuffer->bytesRead = 0;
                        request->readBuffer->bytesToRead =
                            MAXPAYLOAD;
                        readRequest(request);
                        break;
                    case STATE_READ:
                        readRequest(request);
                        break;
                }
            }
            if (httpRRArray[i].state == STATE_SEND &&
                (httpRRArray[i].socket, &write_fds)) {
                printf("write fds\n");

                sendResponse(&httpRRArray[i]);
            }
            if (httpRRArray[i].state == STATE_COMPLETE ||
                httpRRArray[i].state == STATE_ERROR) {
                shutdown(httpRRArray[i].socket, SHUT_RDWR);
                close(httpRRArray[i].socket);
                for (int k = 0; k < N; k++) {
                    if (fdArray[n][k] ==
                        httpRRArray[i].socket)
                        fdArray[n][k] = -1;
                }
            }
        }
    }
}

```

```
    }  
    httpRRArray[i].socket = -1;  
}  
  
}  
  
}
```

Реализация функции отправки ответа клиенту представлена в листинге 3.3.

Листинг 3.3 – Реализация функции отправки ответа клиенту

```
void sendResponse(httpRequestResponsePtr request) {
    request->state = STATE_SEND;

    int bytesSent = send(request->socket,
        request->writeBuffer->data +
            request->writeBuffer->bytesWritten,
        request->writeBuffer->size, 0);

    if (bytesSent == -1) {
        request->errorCode = errno;
        if (errno == EAGAIN || errno == EWOULDBLOCK) {
            return;
        }
        request->state = STATE_ERROR;
        return;
    }

    if (bytesSent > 0) {
        request->writeBuffer->bytesWritten += bytesSent;
        request->writeBuffer->size -= bytesSent;
        if (request->writeBuffer->size == 0) {
            request->state = STATE_COMPLETE;
        }
    }
}
```

Реализация функции создания пула потоков 3.4.

Листинг 3.4 – Реализация функции создания пула потоков

```
void initializeThreads() {
    pthread_t tmpThread;
```

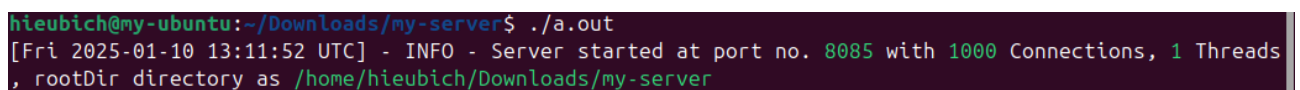
```

threads = malloc(sizeof(pthread_t) * maxThreads);
httpRRArray = malloc(sizeof(httpRequestResponse) *
    maxConnections);
for (int i = 0; i < maxConnections; i++) {
    httpRRArray[i].readBuffer =
        malloc(sizeof(httpReadBuffer));
    httpRRArray[i].writeBuffer =
        malloc(sizeof(httpWriteBuffer));
    // httpRRArray[i].socket = -1; // Initialize socket to -1
}
for (int i = 0; i < maxThreads; i++)
{
    for (int j = 0; j < N; j++)
    {
        fdArray[i][j] = -1;
    }
}
for (int i = 0; i < maxThreads; i++) {
    int *slot = malloc(sizeof(int));
    *slot = i;
    if (pthread_create(&threads[i], NULL, responseLoop,
        slot))
    {
        logMessage(LOG_ERROR, strerror(errno), 0);
    }
}
}
}

```

3.3 Демонстрация работы программы

На рисунке 3.1 представлен пример запуска сервера.



```

hieubich@my-ubuntu:~/Downloads/my-server$ ./a.out
[Fri 2025-01-10 13:11:52 UTC] - INFO - Server started at port no. 8085 with 1000 Connections, 1 Threads
, rootDir directory as /home/hieubich/Downloads/my-server

```

Рисунок 3.1 – Пример запуска сервера

На рисунке 3.2 представлен пример GET-запрос.

```
hieubich@my-ubuntu:~/Downloads/my-server$ curl http://localhost:8085/sometext.txt  
seti2024
```

Рисунок 3.2 – Пример ответа на GET-запрос

На рисунке 3.3 представлен пример ответа для файла (200MB).

```
hieubich@my-ubuntu:~/Downloads/my-server$ curl -I http://localhost:8085/200.zip  
HTTP/1.1 200 OK  
Content-Length: 209715200  
Content-Type: text/plain  
Connection: close
```

Рисунок 3.3 – Пример ответа для файла (200MB)

На рисунке 3.4 представлен пример ответа на HEAD-запрос для Си файла.

```
hieubich@my-ubuntu:~/Downloads/my-server$ curl -I http://localhost:8085/main.c  
HTTP/1.1 200 OK  
Content-Length: 14577  
Content-Type: text/plain  
Connection: close
```

Рисунок 3.4 – Пример ответа на HEAD-запрос

На рисунке 3.5 представлен пример ответа на запрос несуществующего файла.

```
hieubich@my-ubuntu:~/Downloads/my-server$ curl -I http://localhost:8085/1000.zip
HTTP/1.1 404 Not Found
Content-Length: 22
Content-Type: text/html
Connection: close
```

Рисунок 3.5 – Пример ответа на несуществующего файла

На рисунке 3.6 представлен пример ответа на запрещенный запрос.

```
hieubich@my-ubuntu:~/Downloads/my-server$ curl -X POST http://localhost:8085/main.c
<h1>405 Method Not Allowed</h1>hieubich@my-ubuntu:~/Downloads/my-server$ S
```

Рисунок 3.6 – Пример ответа на запрещенный запрос

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- 1) операционная система — Ubuntu 24.04.1 (64-bit);
- 2) объем оперативной памяти — 4 Гбайт;
- 3) процессор — 4 ядер.

В качестве альтернативного сервера для анализа был выбран **nginx** [3], так как он считается одним из самых популярных и востребованных веб-серверов. В качестве инструмента для создания нагрузки и оценки производительности будет применяться **ab** (Apache Benchmark) [4]. Тестирование было направлено на проверку отправки запросов к `main.c`, размер которого 14.6кБ.

4.2 Результаты исследования

На рисунке 4.1 представлены результаты сравнения серверов.

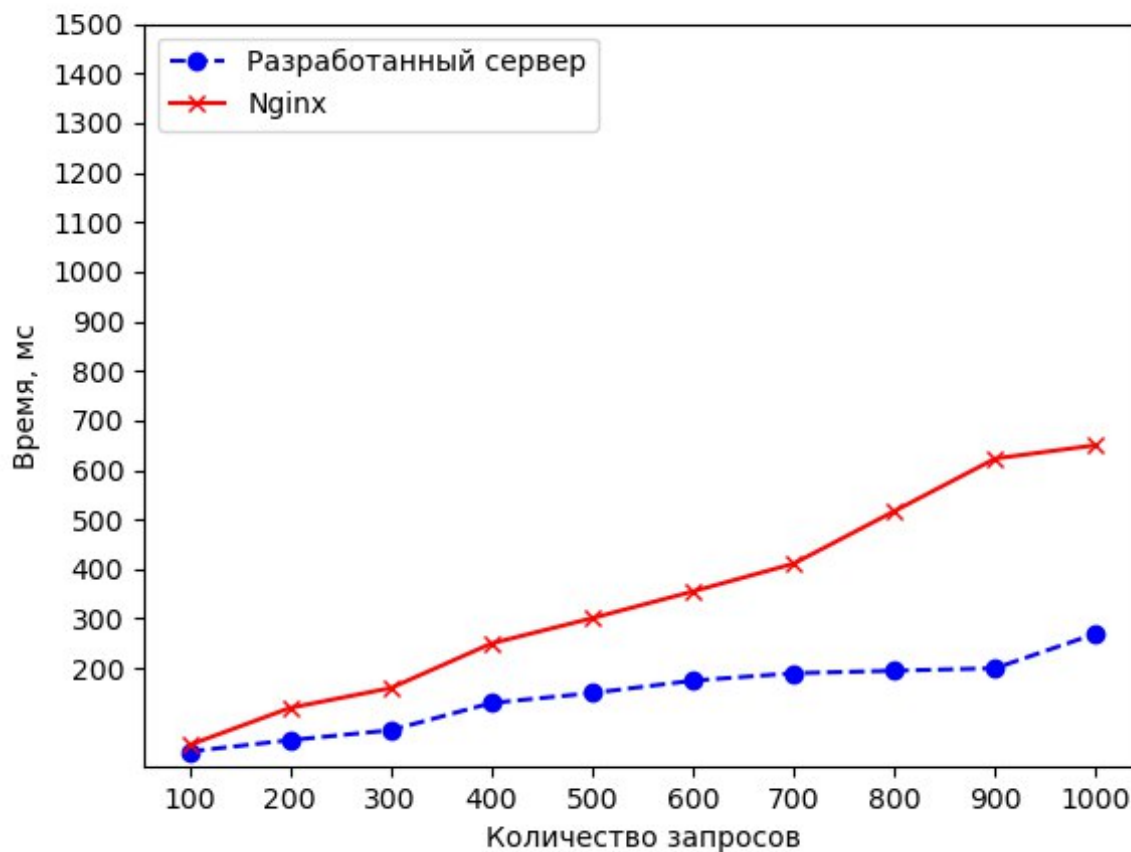


Рисунок 4.1 – Результаты сравнения серверов

Вывод

На графике показано время обработки запросов для разработанного сервера и сервера Nginx в зависимости от количества запросов. Разработанный сервер характеризуется более стабильной производительностью, время обработки запросов практически не меняется при увеличении нагрузки. Разница во времени обработки становится более заметной при большом количестве запросов. Например, при 1000 запросах разработанный сервер показывает результат в 2.13 раз быстрее, в то время как при 200 запросов эта разница 1.17.

ЗАКЛЮЧЕНИЕ

Цель работы, заключающаяся в разработке сервера для отдачи статического содержимого с диска с использованием паттерна thread-pool совместно с системным вызовом pselect, была достигнута.

Были решены следующие задачи:

- 1) проанализирована архитектура «пул потоков» и системный вызов pselect;
- 2) спроектирован алгоритм работы сервера;
- 3) разработан сервер раздачи статических данных;
- 4) приведено сравнение разработанного сервера с nginx.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Yibei Ling, Tracy Mullen, Xiaola Lin*. Analysis of Optimal Thread Pool Size. — 2000.
2. *Керниган Б. В.* Язык программирования С, 2-е издание. — Издательский дом Вильямс, 2012. — С. 25.
3. *Reese W.* Nginx: the high-performance web server and reverse proxy // Linux Journal. — 2008. — Т. 2008, № 173. — С. 2.
4. Apache Benchmark. — [Электронный ресурс]. — Режим доступа: <https://httpd.apache.org/docs/2.4/programs/ab.html> (дата обращения: 12.12.24).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе состоит из 9 слайдов