

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего  
образования

Электротехнический университет

Кафедра Информационные технологии и автоматизированные системы

### Системное программирование

#### Курсовая работа

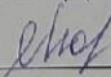
Тема: «Взаимодействие процессов в многозадачной среде»

Вариант 31

Выполнила:

студентка группы РИС-19-16

Люкина Д.С.

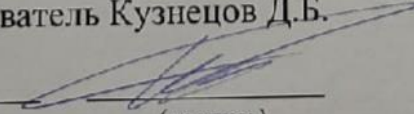


Проверил:

ст. преподаватель Кузнецов Д.Б.



(оценка)

  
(подпись)

28.12.2022

Пермь, 2021

## СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ.....</b>	<b>3</b>
<b>1. Анализ предметной области .....</b>	<b>4</b>
1.1. Постановка задачи .....	4
1.2. Анализ задачи.....	4
1.3. Инструменты разработки.....	5
<b>2. Практическая часть .....</b>	<b>7</b>
2.1. Алгоритм работы клиентской части .....	7
2.2. Организация многозадачности сервера.....	9
2.3. Сценарий компиляции.....	9
2.4. Демонстрация работы программы .....	10
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>14</b>

## ВВЕДЕНИЕ

Системное программирование — это практика написания системного ПО, низкоуровневый код которого взаимодействует непосредственно с ядром и основными системными библиотеками.[1] Оно включает в себя разработку отдельных частей программного обеспечения, которые позволяют всей системе функционировать как единое целое.

Одной из составляющей системного программирования являются клиент-серверные приложения. Клиент-серверное приложение состоит из клиентской программы, которая использует услуги, предоставляемые серверной программой. Социальные сети (Фейсбук, ВК и пр.), сайты электронной коммерции (Amazon, Озон и др.), мобильные приложения (Instagram и т.д.), устройства Интернета вещей (умные колонки или смарт-часы) работают на основе клиент-серверной архитектуры. Поэтому данная тема довольно актуальна в наше время.

**Цель** курсовой работы: разработка программы на Си из двух частей: клиент и сервер, в соответствии с заданием варианта.

Для достижения цели необходимо решить следующие **задачи**:

1. провести анализ предметной области
2. выбрать инструменты для реализации
3. разработать программу
4. продемонстрировать работу программы

## **1. Анализ предметной области**

### **1.1. Постановка задачи**

Необходимо разработать программу на Си. Обмен между клиентом и сервером должен происходить с использованием именованных каналов. При этом каждое обращение клиента сервер должен обрабатывать в отдельном процессе.

В ответ сервер должен передать клиенту номер запроса клиента, начиная от старта сервера.

### **1.2. Анализ задачи**

Создание клиент-серверного приложения подразумевает разработку отдельно серверной и клиентской части так, чтобы можно было запустить один сервер, к которому подключается и отключаются клиенты, запуск которых производится из терминала. Один сервер должен поддерживать параллельную работу со многими клиентами.

На каждого клиента необходимо создание нового процесса, поэтому сервер ожидает подключения клиента, и если оно происходит успешно, порождает новый процесс, который продолжает ожидание клиента.

Для того, чтобы сервер мог возвращать номер запроса, в файле сервера добавляется счетчик, который обновляется при получении новых запросов с самого начала работы сервера.

Общение между клиентом и сервером реализовано при помощи именованных каналов, которые являются механизмом межпроцессного взаимодействия (IPC), предоставляющего канал связи для дескриптора файла. Доступ к именованному каналу выполняется через специальный файл. Обычные конвейеры применяются именно для того, чтобы «перекачивать» вывод одной программы во ввод другой; они создаются в памяти посредством системного вызова и не существуют в какой-либо файловой

системе. Именованные каналы действуют как и обычные, но обращение к ним происходит через файл, называемый специальным файлом FIFO. Несвязанные процессы могут обращаться к этому файлу и обмениваться информацией.[1] Для закрытия именованных каналов используются сигналы - средство уведомления процесса о наступлении некоторого события в системе. Инициатором отправки сигнала может выступать как другой процесс, так и сама ОС. Сигналы, посылаемые ОС, уведомляют о наступлении некоторых строго определенных ситуаций. В случае прерывания работы процесса применяется комбинация клавиш Ctrl+C.

### 1.3. Инструменты разработки

**Cи** – компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах Деннисом Ритчи. Первоначально он был разработан для реализации операционной системы UNIX, но, впоследствии, был перенесён на множество других платформ. Язык программирования Си оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования, как C++, C#, Java и Objective-C. Этот язык выбран, потому что он подходит для функционального программирования.

**Linux** - семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты. [2] Разработка приложения ведется в Ubuntu (при помощи Oracle VM VirtualBox). Выбран по требованию в варианте.

Выбрана среда разработки **Sublime Text 3**, поскольку подходит для разработки под Linux и поддерживает разработку на языке C.

**Makefile** - файл, содержащий набор инструкций для программы **make**. Программа make с помощью этого файла позволяет автоматизировать процесс компиляции программы и выполнять при этом различные действия.

При запуске `make` по умолчанию ищет файл `Makefile` в текущей папке и обрабатывает. Система `make` родилась в мире UNIX и постепенно переползла и на Windows вместе с портами GNU-компиляторов (**gcc**).

## 2. Практическая часть

### 2.1. Алгоритм работы клиентской части

Рассмотрим алгоритм работы клиентской части приложения (рисунок

1)

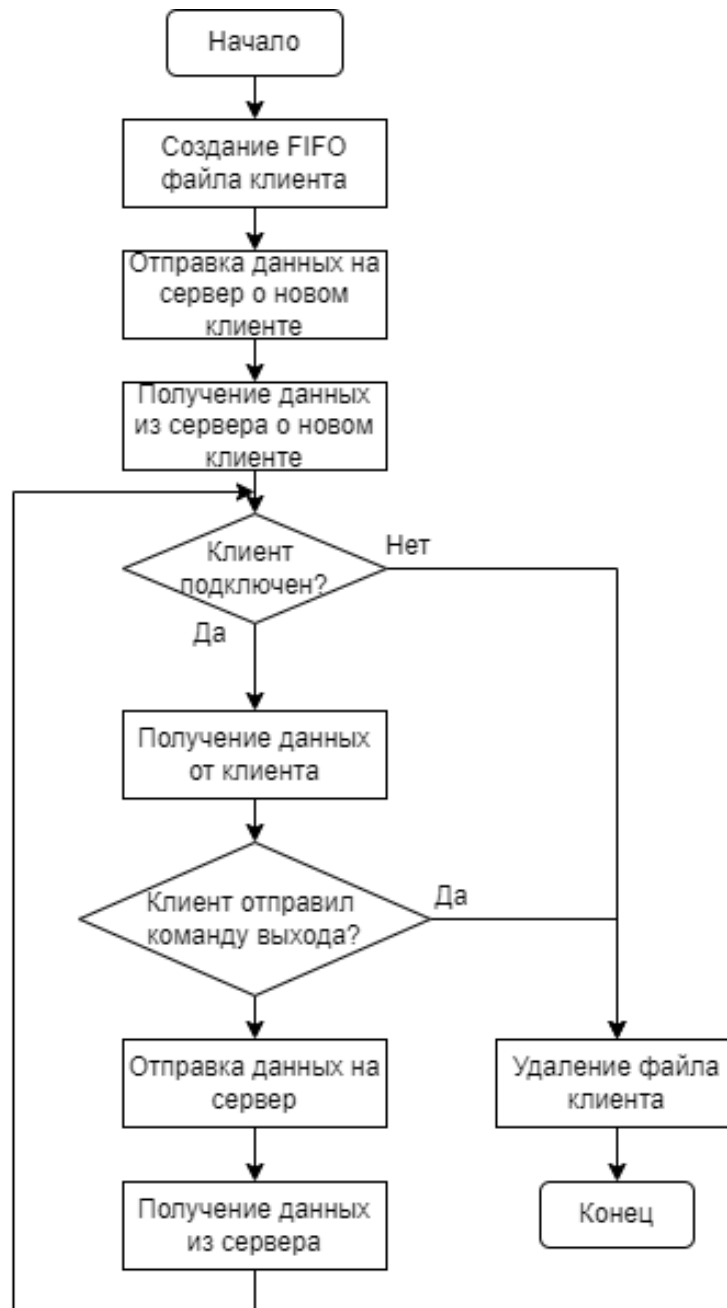


Рисунок 1 – алгоритм работы клиентской части

Клиентская часть выполняет следующие задачи:

- 1) Программа начинает свою работу с присвоения клиенту своего уникального идентификатора, который соответствует идентификатору процесса.

- 2) Создается FIFO файл – индивидуальный именованный канал для клиента.
- 3) Ввод сообщений клиентом в бесконечном цикле. При написании пользователем слова «exit» канал удаляется, уведомление об этом отсылается на сервер.

Рассмотрим алгоритм работы серверной части приложения (рисунок 2).

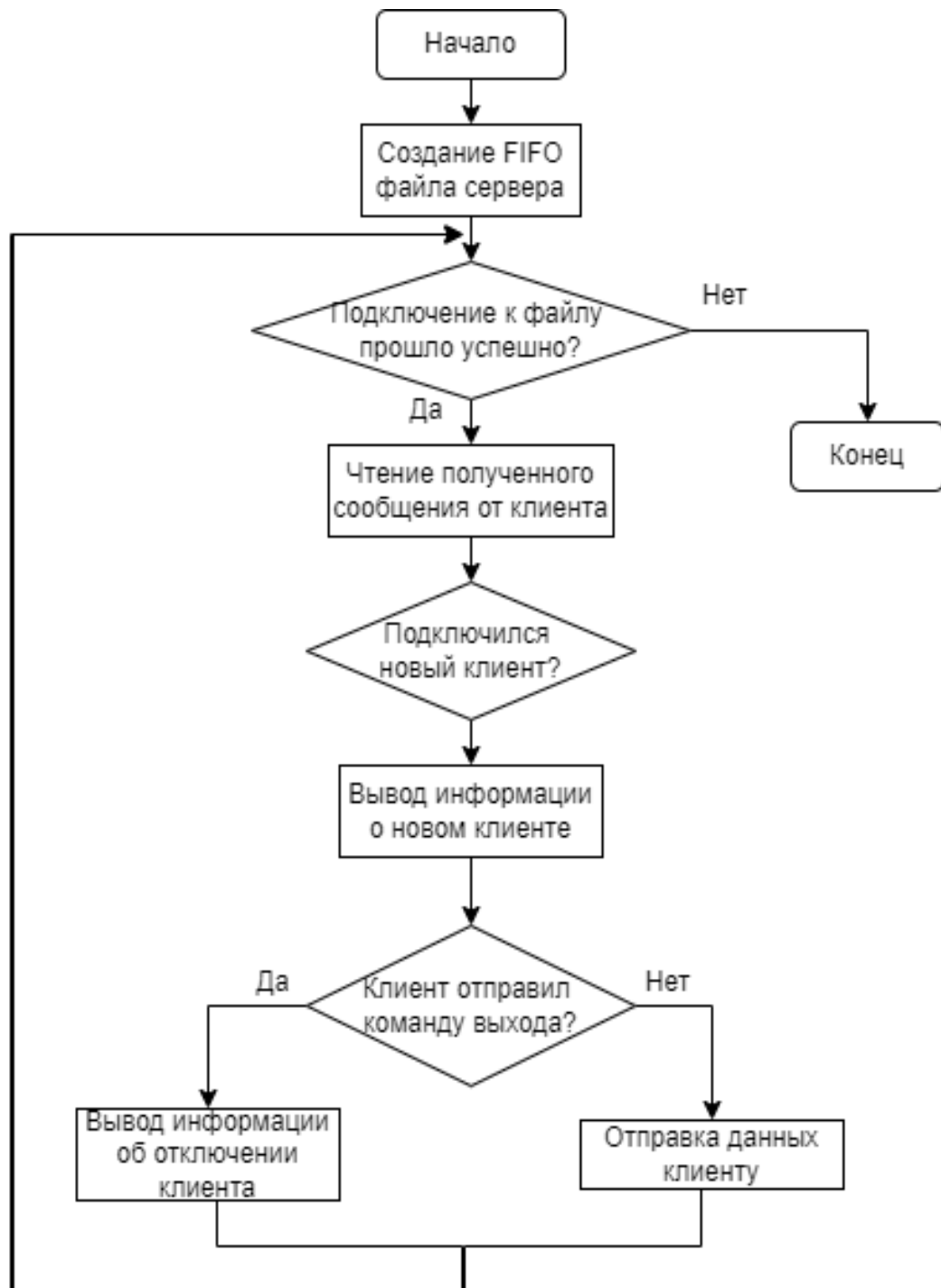


Рисунок 2 – Алгоритм работы серверной части приложения



Серверная часть выполняет следующие пункты:

- 1) Объявляется счетчик запросов.
- 2) Проверка - не был ли подан сигнал SIGINT. В случае положительного результата, сервер прекращает работу.
- 3) Создание FIFO файла сервера.
- 4) Работа бесконечного цикла:
  - 4.1. Проверка открытия файла сервера
  - 4.2. Получение сообщений от клиента
  - 4.3. Проверка подключения нового клиента, сообщение об этом на сервер
  - 4.4. Присвоение клиенту номера запроса и увеличение счетчика запросов на 1
  - 4.5. Данные отправляются клиенту, если он не отключен
  - 4.6. Если отправленный текст и полученный соответствуют друг другу, то приходит сообщение об успешности операции

Из всего выше сказанного, можно сказать, что сервер выполняется для записи данных о клиентах, и уведомление об этом самих клиентов.

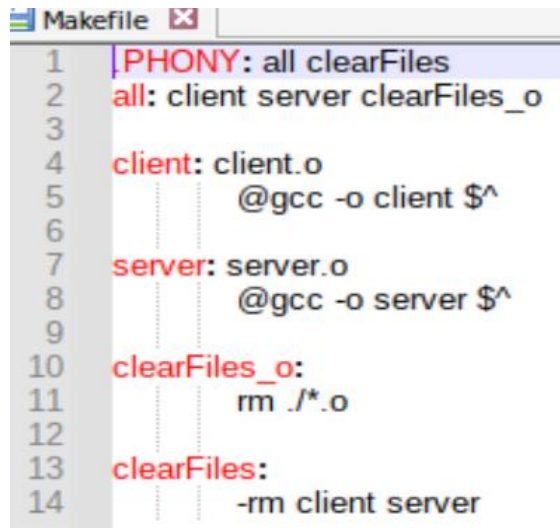
## **2.2. Организация многозадачности сервера**

Когда соединение клиента и сервера подтверждено, то создается новый процесс, который будет работать с клиентом напрямую, в то время как родительский процесс продолжает ожидание новых клиентов, и если такие находятся, то схема повторяется. Как только процесс заканчивает работу с клиентом, то его работа завершается, и сигнал об этом отправляется родительскому процессу.

## **2.3. Сценарий компиляции**

Курсовой проект состоит из трёх файлов: клиента, сервера и общего файла, содержащий библиотеки и переменные, которые используют сразу оба файла, названные первыми.

Компиляция файлов происходит с помощью Makefile (рисунок 3).



```

1 | PHONY: all clearFiles
2 | all: client server clearFiles_o
3 |
4 | client: client.o
5 |     @gcc -o client $^
6 |
7 | server: server.o
8 |     @gcc -o server $^
9 |
10 | clearFiles_o:
11 |     rm ./*.o
12 |
13 | clearFiles:
14 |     -rm client server

```

Рисунок 3 – содержание Makefile

Для запуска Makefile необходимо в терминале использовать команду `make`, находясь в соответствующем каталоге. Для очистки использовать `make clearFiles`.

## 2.4. Демонстрация работы программы

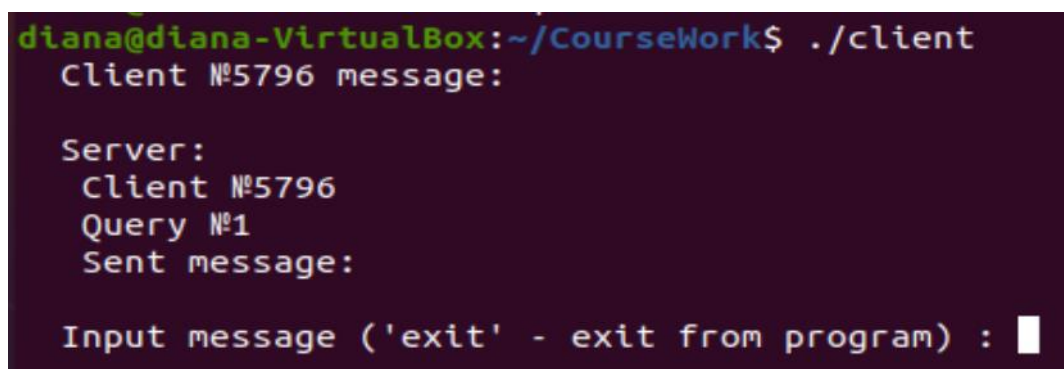
Листинг программ располагается в Листинге А.

Запуск программы сервера выполняется командой:

```
./server
```

Запуск программы клиента выполняется командой (рисунок 4):

```
./client
```



```

diana@diana-VirtualBox:~/CourseWork$ ./client
Client №5796 message:

Server:
Client №5796
Query №1
Sent message:

Input message ('exit' - exit from program) : 

```

Рисунок 4 – Запуск программы клиента

На экран выводится информация, которую получил сервер при подключении клиента: это его ID, номер запроса и отправленное сообщение. Но из-за того, чтобы клиент только зашел и далее никак больше не взаимодействовал с программой, серверу передалось сообщение без

содержания. После этого клиенту предлагается ввести сообщение, а также прилагается информация, как выйти из программы.

На сервер же поступает уведомление, что появился новый клиент и вывод его данных (рисунок 5).

```
diana@diana-VirtualBox:~/CourseWork$ ./server  
  
-> New client №5796  
Client №5796 message:  
Query №1  
Message delivered!
```

Рисунок 5 – работа сервера

И каждый раз, когда клиент вводит новое сообщения (Рисунок 6), все данные отправляются на сервер, включая номер запроса клиента (Рисунок 7).

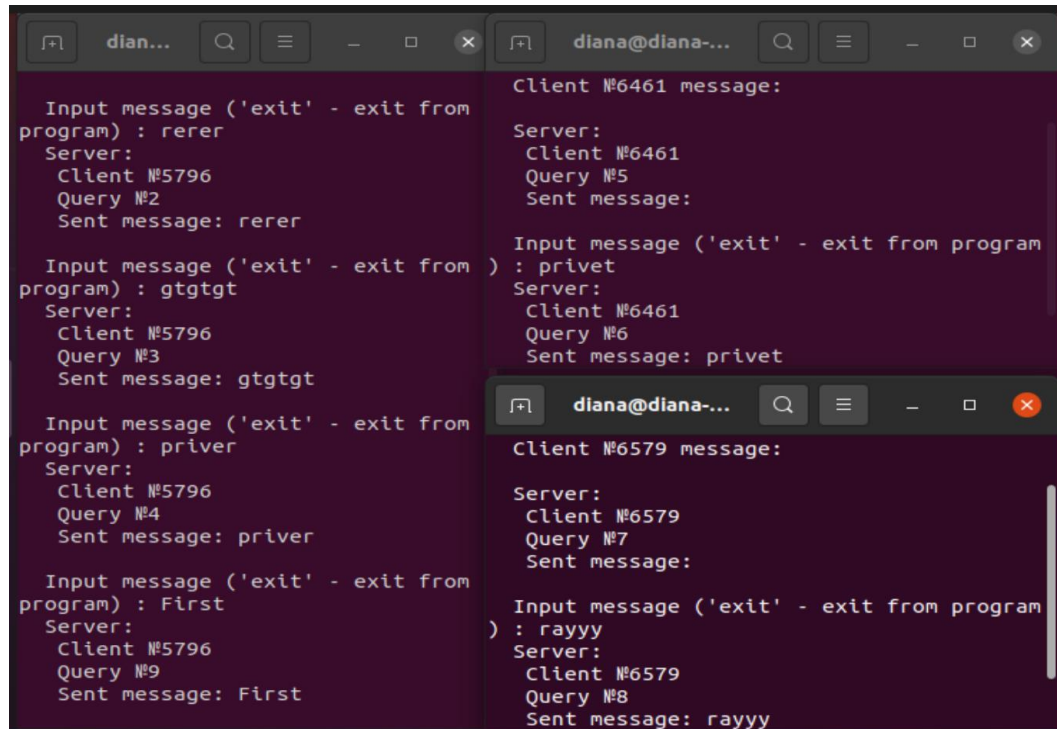
```
Server:  
Client №5796  
Query №1  
Sent message:  
  
Input message ('exit' - exit from program) : rerer  
Server:  
Client №5796  
Query №2  
Sent message: rerer  
  
Input message ('exit' - exit from program) : gtgtgt  
Server:  
Client №5796  
Query №3  
Sent message: gtgtgt  
  
Input message ('exit' - exit from program) : priver  
Server:  
Client №5796  
Query №4  
Sent message: priver  
  
Input message ('exit' - exit from program) : █
```

Рисунок 6 – взаимодействие пользователя с программой

```
-> New client №5796  
Client №5796 message:  
Query №1  
Message delivered!  
  
Client №5796 message: rerer  
Query №2  
Message delivered!  
  
Client №5796 message: gtgtgt  
Query №3  
Message delivered!  
  
Client №5796 message: priver  
Query №4  
Message delivered!
```

Рисунок 7 – отображение действий клиента на сервере

Для примера введем ещё двоих пользователей. Отображение работы клиентов и работы сервера изображены на рисунках 8 и 9 соответственно.



The image shows two terminal windows. The left window, titled 'dian...', displays the server's perspective of interactions with three clients. It shows input messages from clients, the server's response, and the client's acknowledgment. The right window, titled 'diana@diana-...', shows the client's perspective for three different clients (№6461, №6579, and №6579), displaying their input messages, the server's response, and the client's acknowledgment.

```
dian... Client №5796 message:
Input message ('exit' - exit from program) : rerer
Server:
Client №5796
Query №2
Sent message: rerer

Input message ('exit' - exit from program) : gtgtgt
Server:
Client №5796
Query №3
Sent message: gtgtgt

Input message ('exit' - exit from program) : priver
Server:
Client №5796
Query №4
Sent message: priver

Input message ('exit' - exit from program) : First
Server:
Client №5796
Query №9
Sent message: First

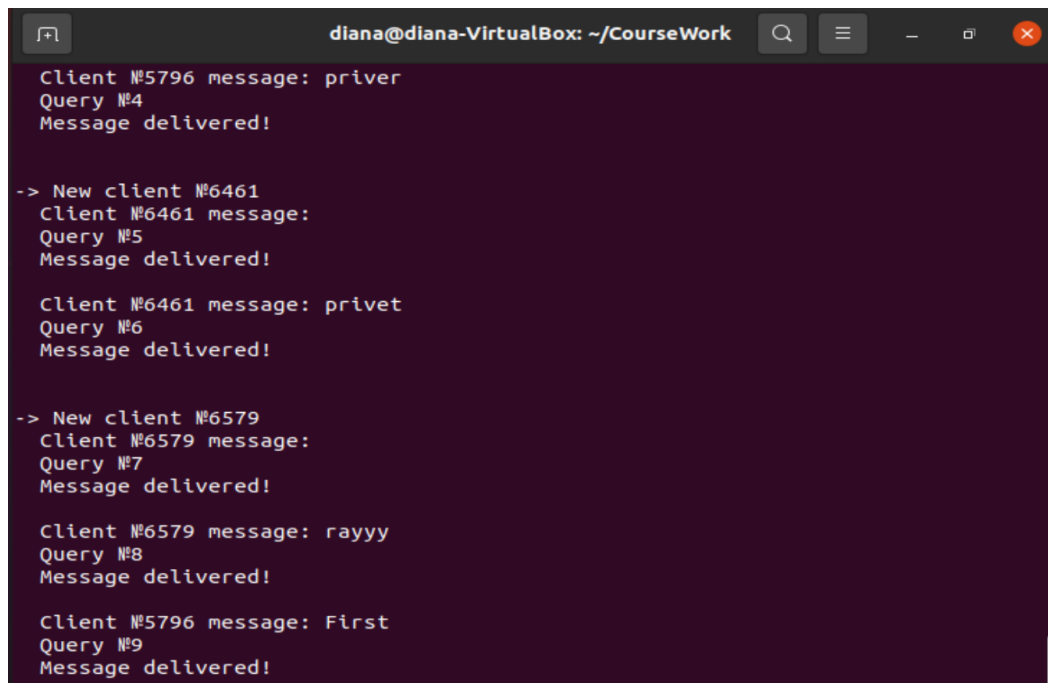
diana@diana-... Client №6461 message:
Server:
Client №6461
Query №5
Sent message:

Input message ('exit' - exit from program) : privet
Server:
Client №6461
Query №6
Sent message: privet

diana@diana-... Client №6579 message:
Server:
Client №6579
Query №7
Sent message:

Input message ('exit' - exit from program) : rayyy
Server:
Client №6579
Query №8
Sent message: rayyy
```

Рисунок 8 – взаимодействие с программой трех клиентов



The image shows a terminal window titled 'diana@diana-VirtualBox: ~/CourseWork'. It displays the server's perspective of interactions with three clients. It shows the server receiving messages from clients, processing them, and sending responses. The output includes the client's message, the server's response, and the client's acknowledgment.

```
diana@diana-VirtualBox: ~/CourseWork Client №5796 message: priver
Query №4
Message delivered!

-> New client №6461
Client №6461 message:
Query №5
Message delivered!

Client №6461 message: privet
Query №6
Message delivered!

-> New client №6579
Client №6579 message:
Query №7
Message delivered!

Client №6579 message: rayyy
Query №8
Message delivered!

Client №5796 message: First
Query №9
Message delivered!
```

Рисунок 9 – отображение процесса сервера при взаимодействии с программой сразу трех клиентов

Выход клиента из программы осуществляется вводом слова “exit” (рисунок 10).

```
Input message ('exit' - exit from program) : exit
Client №6461 came out
diana@diana-VirtualBox:~/CourseWork$
```

Рисунок 10 – выход пользователя из программы

Уведомление об отключении клиента поступает на сервер (рисунок 11).

```
Client №6461 message: exit
Query №10
Client №6461 came out
```

Рисунок 11 – уведомление на сервере о выходе клиента из программы

Выход из сервера осуществляется посредством нажатия клавиш Ctrl+C (рисунок 12).

```
^C
Server closed
diana@diana-VirtualBox:~/CourseWork$
```

Рисунок 12 – Окончание работы сервера

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы были выполнены все задачи: был проведен анализ предметной области, были составлены алгоритмы работы программы, выбраны инструменты разработки и разработан программный продукт.

Цель курсовой работы достигнута - разработана программа на Си, в которой обмен между клиентом и сервером происходит с использованием именованных каналов, а при этом каждое обращение клиента сервер обрабатывает его в отдельном процессе.

За время выполнения курсовой работы, был получен опыт в разработке приложений, работающих в многозадачной среде.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Лав Р. Linux. Системное программирование. СПб.: Питер, 2014. 448 с.
2. Бикманс Ж. Linux с нуля. Москва: ДМК-Пресс, 2016. 428 с.
3. МакГрат М. Программирование на С для начинающих. Москва: Эксмо, 2016. 192 с.

## ПРИЛОЖЕНИЕ

### Приложение А

#### Листинг А.1 – Код файла general.h

```
#include <stdio.h>
#include <locale.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>

#define PUB_FIFO "Server_NamedPipe"
#define PRIV_FIFO "Client_NamedPipe_"
#define NEW_CLIENT " \n"
#define EXIT_FROM_PROGRAM "exit\n"

struct NamedPipe_Data{
    int ID_client;
    char message[100];
};

struct NamedPipe_Data Info_to_Server, Info_to_Client;
char Priv_Name[20];
int PubFd, PrivFd;
char* Priv_NamedPipe_Name;
```

#### Листинг А.2 – Код файла server.c

```
#include "general.h"

int numbur_query = 1; /* Счетчик запросов */

/*Отключение сервера при помощи сигнала SIGINT*/
void catchSignal(int num){
    unlink(PUB_FIFO);
    printf("\nServer closed\n");
    exit(0);
}

int main(){
    setlocale(LC_ALL, "Rus");

    /* Проверка на сигнал SIGINT*/
    if(signal(SIGINT, &catchSignal) == SIG_ERR){
        exit(1);
    }

    /* Создание канала */
    int TempFd;
    if((TempFd = open(PUB_FIFO, O_RDONLY)) == -1){
        umask(0);
        mknod(PUB_FIFO, S_IFIFO|0666, 0);
    }
}
```



```

    }
    else{
        close(TempFd);
    }

    while(1){

        if((PubFd = open(PUB_FIFO, O_RDONLY)) < 0){
            printf("ERROR (PUBLIC_FIFO)\n");
            exit(1);
        }

        /* Вывод данных о клиенте */
        if(read(PubFd, &Info_to_Server, sizeof(struct
NamedPipe_Data)) > 0){
            char TempBuffer[6];
            strcpy(Priv_Name, PRIV_FIFO);
            sprintf(TempBuffer, "%d", Info_to_Server.ID_client);
            strcat(Priv_Name, TempBuffer);
            Priv_NamedPipe_Name = Priv_Name;

            /* Создание файла для нового клиента */
            if(strcmp(Info_to_Server.message, NEW_CLIENT) == 0){
                printf("\n->      New      client      №%d\n",
Info_to_Server.ID_client);

                /* Создание канала */
                if((TempFd = open(Priv_NamedPipe_Name, O_RDONLY))
== -1){
                    umask(0);
                    mknod(Priv_NamedPipe_Name, S_IFIFO|0666, 0);
                }
                else{
                    close(TempFd);
                }

                printf("      Client      №%d      message:      %s",
Info_to_Server.ID_client, Info_to_Server.message);
                printf("  Query №%d\n", numbur_query);

                /* Проверка на отключение клиента */
                if(strcmp(Info_to_Server.message, EXIT_FROM_PROGRAM)
== 0){
                    unlink(Priv_NamedPipe_Name);
                    printf("Client      №%d      came      out\n\n",
Info_to_Server.ID_client);
                }
                else{
                    /* Ответ от сервера клиенту */
                    if((PrivFd = open(Priv_NamedPipe_Name, O_WRONLY))
> 0){
                        Info_to_Client.ID_client =
Info_to_Server.ID_client;
                        sprintf(Info_to_Client.message, "      Client
№%d\n  Query №%d\n  Sent message: ", \
Info_to_Server.ID_client, numbur_query);
                        numbur_query+=1;

```

```

        strcat(Info_to_Client.message,
Info_to_Server.message);
        if(write(PrivFd,          &Info_to_Client,
sizeof(struct NamedPipe_Data))){
            printf("  Message delivered!\n\n");
            close(PrivFd);
        }
    }
}

}
else{
printf("ERROR - connection!\n");
exit(1);
}
}
return 0;
}

```

### Листинг А.3 – Код файла client.c

```

#include "general.h"

int main(){
    setlocale(LC_ALL, "Rus");

    /* Первое подключение к серверу */
    Info_to_Server.ID_client = getpid();
    strcpy(Info_to_Server.message, NEW_CLIENT);

    /* Получение имени для файла */
    char TempBuffer[6];
    strcpy(Priv_Name, PRIV_FIFO);
    sprintf(TempBuffer, "%d", getpid());
    strcat(Priv_Name, TempBuffer);

    Priv_NamedPipe_Name = Priv_Name;
    printf("  Client №%d message: %s\n", \
Info_to_Server.ID_client, Info_to_Server.message);

    /* Отправка сообщений серверу и принятие*/
    if((PubFd = open(PUB_FIFO, O_WRONLY)) > 0){
        if(write(PubFd,          &Info_to_Server,          sizeof(struct
NamedPipe_Data)) > 0){
            close(PubFd);
        }
        else{
            printf("ERROR - writing text\n");
        }
        usleep(200000);

        /* Клиент отключился */
        if(strcmp(Info_to_Server.message, EXIT_FROM_PROGRAM) == 0){
            printf("          Client          №%d          came          out\n",
Info_to_Server.ID_client);
            exit(0);
        }
    }
}

```

```

        /* Получение данных с сервера */
        if((PrivFd = open(Priv_NamedPipe_Name, O_RDONLY)) > 0){
            if(read(PrivFd, &Info_to_Client, sizeof(struct
NamedPipe_Data)) > 0){
                printf("                Server:\n%s\n",
Info_to_Client.message);
                close(PrivFd);
            }
        }
    }
    else{
        printf("ERROR (PUBLIC_FIFO)\n");
        exit(1);
    }

    /* Цикл для продолжения запросов к серверу */
    while(1){
        printf("  Input message ('exit' - exit from program) : ");
        fgets(Info_to_Server.message, 60, stdin);
        Info_to_Server.ID_client = getpid();

        /* Отправка сообщений серверу и принятие */
        if((PubFd = open(PUB_FIFO, O_WRONLY)) > 0){
            if(write(PubFd, &Info_to_Server, sizeof(struct
NamedPipe_Data)) > 0){
                close(PubFd);
            }
            else{
                printf("ERROR - writing text\n");
            }
            usleep(200000);

            /* Клиент отключился */
            if(strcmp(Info_to_Server.message, EXIT_FROM_PROGRAM)
== 0){
                printf("                Client   №%d   came   out\n",
Info_to_Server.ID_client);
                exit(0);
            }

            /* Получение данных с сервера */
            if((PrivFd = open(Priv_NamedPipe_Name, O_RDONLY)) >
0){
                if(read(PrivFd, &Info_to_Client, sizeof(struct
NamedPipe_Data)) > 0){
                    printf("                Server:\n%s\n",
Info_to_Client.message);
                    close(PrivFd);
                }
            }
            else{
                printf("ERROR (PUBLIC_FIFO)\n");
                exit(1);
            }
        }
    }
}

```