

дисциплина: *Операционные системы*

Студент: Губина Ольга Вячеславовна

Группа: НПИбд-01-20

Преподаватель: Велиева Татьяна Рефатовна

МОСКВА

2021 г.

Цель работы:

Приобретение практических навыков работы с именованными каналами

Задачи:

- 1. Модернизировать предложенные коды;
- 2. Освоить функции `sleep()` и `clock()`.

Теоретическое введение:

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому.

В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий:

- общеоуниковные (именованные каналы, сигналы);
- System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры)
- BSD (сокеты).

Для передачи данных между неродственными процессами можно использовать механизм именованных каналов (named pipes). Данные передаются по принципу *FIFO* (First In First Out) (первым записан — первым прочитан), поэтому они называются также FIFO pipes или просто FIFO. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

Файлы именованных каналов создаются функцией `mkfifo(3)`.

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

Первый параметр — имя файла, идентифицирующего канал, второй параметр — маска прав доступа к файлу.

Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`):

```
mkfifo(FIFO_NAME, 0600)
```

Подробнее с данным типом каталогов можно ознакомиться в статье *"Каналы FIFO"*<sup>[1]</sup>.

Задание:

Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения:

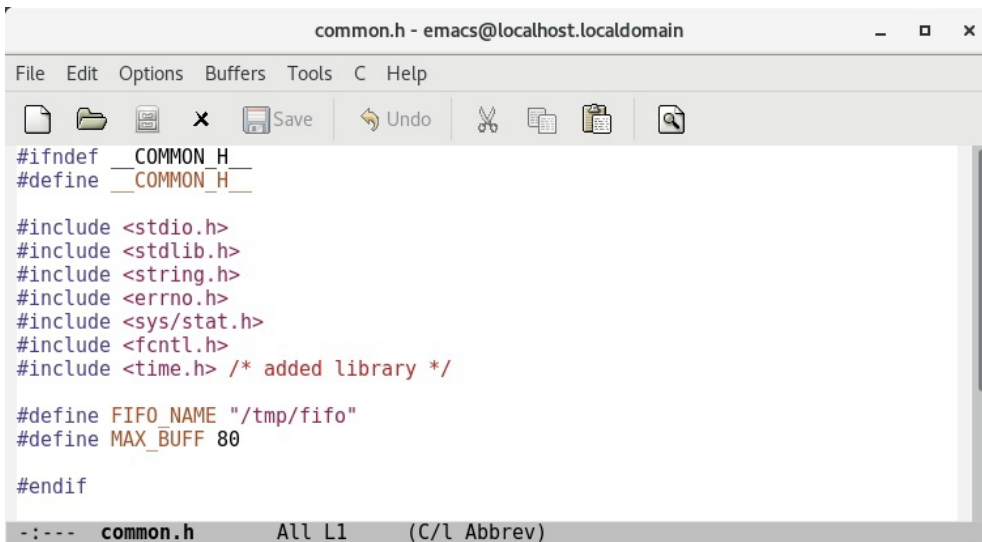
- 1. Работает не 1 клиент, а несколько (например, два).
- 2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию `sleep()` для приостановки работы клиента.
- 3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию `clock()` для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

Выполнение работы:

- 1. Создадим предложенные в лабораторной работе файлы с кодами при помощи текстового редактора `emacs` (рисунок 1). Таким образом, создаем файлы `common.h` (рисунок 2), `server.c` (рисунок), `client.c` (рисунок) и `Makefile` (рисунок) с внесенными в них коррективками, как того от нас требуют задания.

```
[ovgubina@localhost ~]$ cd
[ovgubina@localhost ~]$ emacs common.h
[ovgubina@localhost ~]$ emacs server.c
[ovgubina@localhost ~]$ emacs client.c
[ovgubina@localhost ~]$ emacs Makefile
[ovgubina@localhost ~]$ gcc -c server.c
[ovgubina@localhost ~]$ gcc -o server server.c
[ovgubina@localhost ~]$ gcc -c client.c
[ovgubina@localhost ~]$ gcc -o client client.c
[ovgubina@localhost ~]$
```

рисунок 1: создание файлов и их компиляция



```
common.h - emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
Save Undo
#ifndef COMMON_H
#define COMMON_H

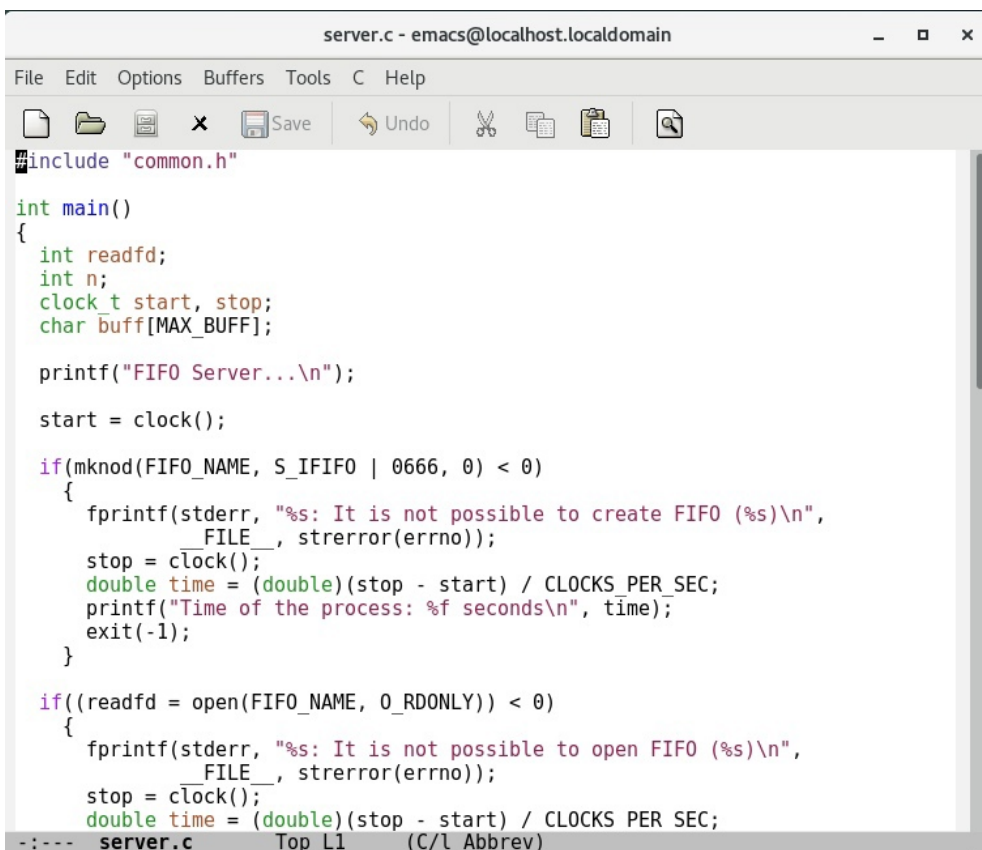
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h> /* added library */

#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80

#endif

-:--- common.h All L1 (C/l Abbrev)
```

рисунок 2: файл common.h



```
server.c - emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
Save Undo
#include "common.h"

int main()
{
    int readfd;
    int n;
    clock_t start, stop;
    char buff[MAX_BUFF];

    printf("FIFO Server...\n");

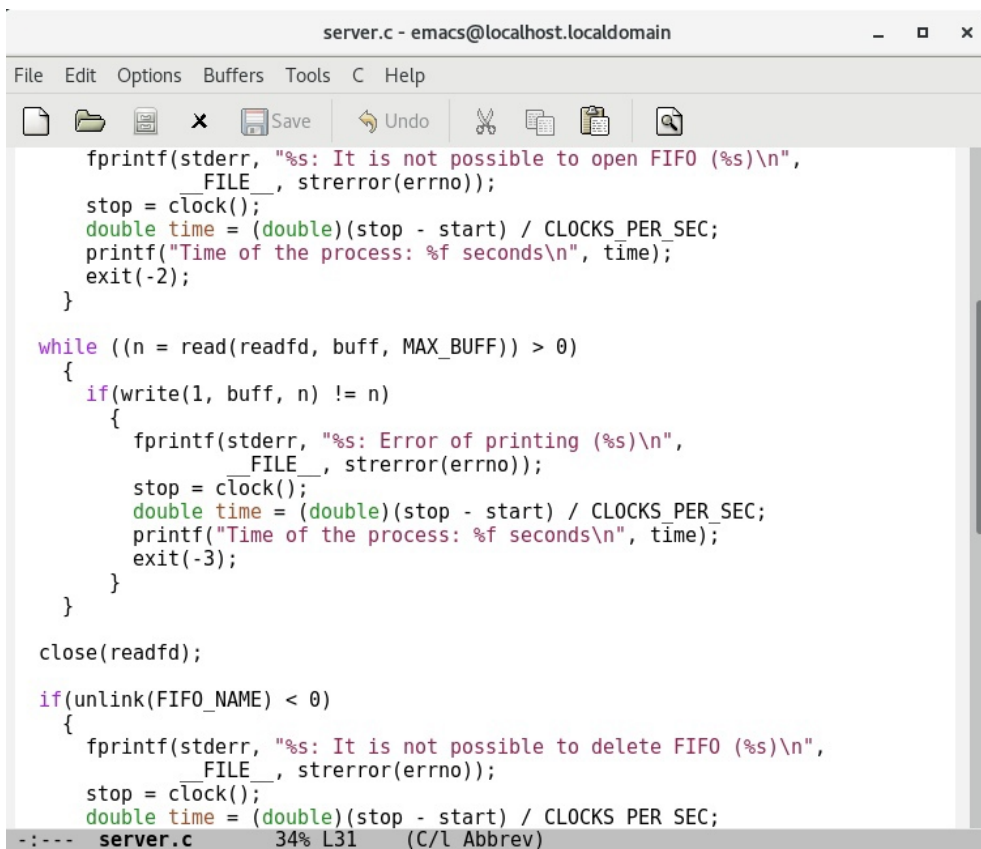
    start = clock();

    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: It is not possible to create FIFO (%s)\n",
            FILE_, strerror(errno));
        stop = clock();
        double time = (double)(stop - start) / CLOCKS_PER_SEC;
        printf("Time of the process: %f seconds\n", time);
        exit(-1);
    }

    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
        fprintf(stderr, "%s: It is not possible to open FIFO (%s)\n",
            FILE_, strerror(errno));
        stop = clock();
        double time = (double)(stop - start) / CLOCKS_PER_SEC;
    }

-:--- server.c Top L1 (C/l Abbrev)
```

рисунок 3: файл server.c



```
server.c - emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]

fprintf(stderr, "%s: It is not possible to open FIFO (%s)\n",
        __FILE__, strerror(errno));
stop = clock();
double time = (double)(stop - start) / CLOCKS_PER_SEC;
printf("Time of the process: %f seconds\n", time);
exit(-2);
}

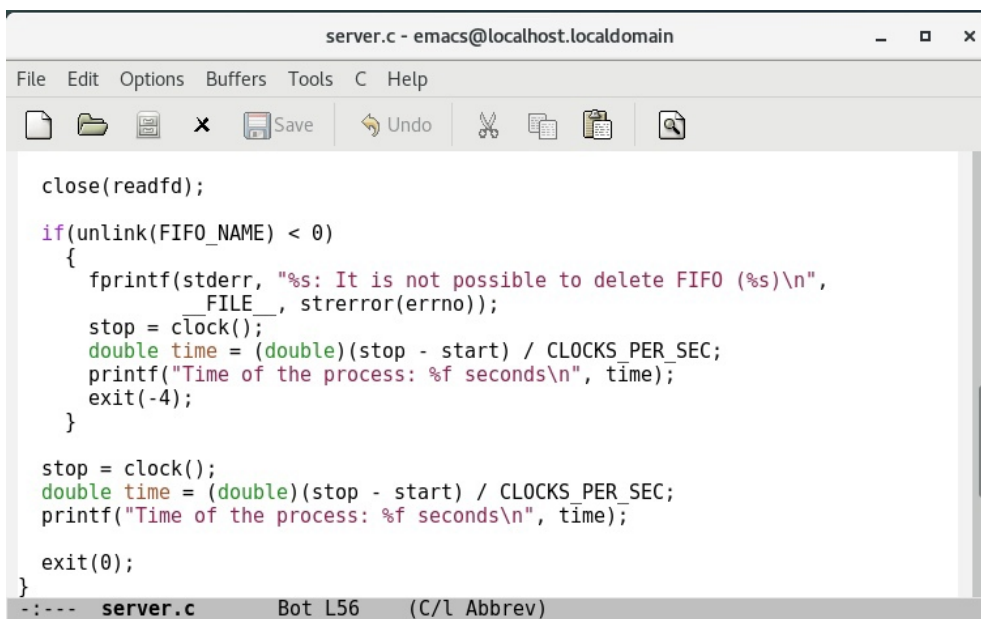
while ((n = read(readfd, buff, MAX_BUFF)) > 0)
{
    if(write(1, buff, n) != n)
    {
        fprintf(stderr, "%s: Error of printing (%s)\n",
                __FILE__, strerror(errno));
        stop = clock();
        double time = (double)(stop - start) / CLOCKS_PER_SEC;
        printf("Time of the process: %f seconds\n", time);
        exit(-3);
    }
}

close(readfd);

if(unlink(FIFO_NAME) < 0)
{
    fprintf(stderr, "%s: It is not possible to delete FIFO (%s)\n",
            __FILE__, strerror(errno));
    stop = clock();
    double time = (double)(stop - start) / CLOCKS_PER_SEC;
    printf("Time of the process: %f seconds\n", time);
    exit(-4);
}

--:--- server.c      34% L31      (C/l Abbrev)
```

рисунок 4: файл server.c



```
server.c - emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]

close(readfd);

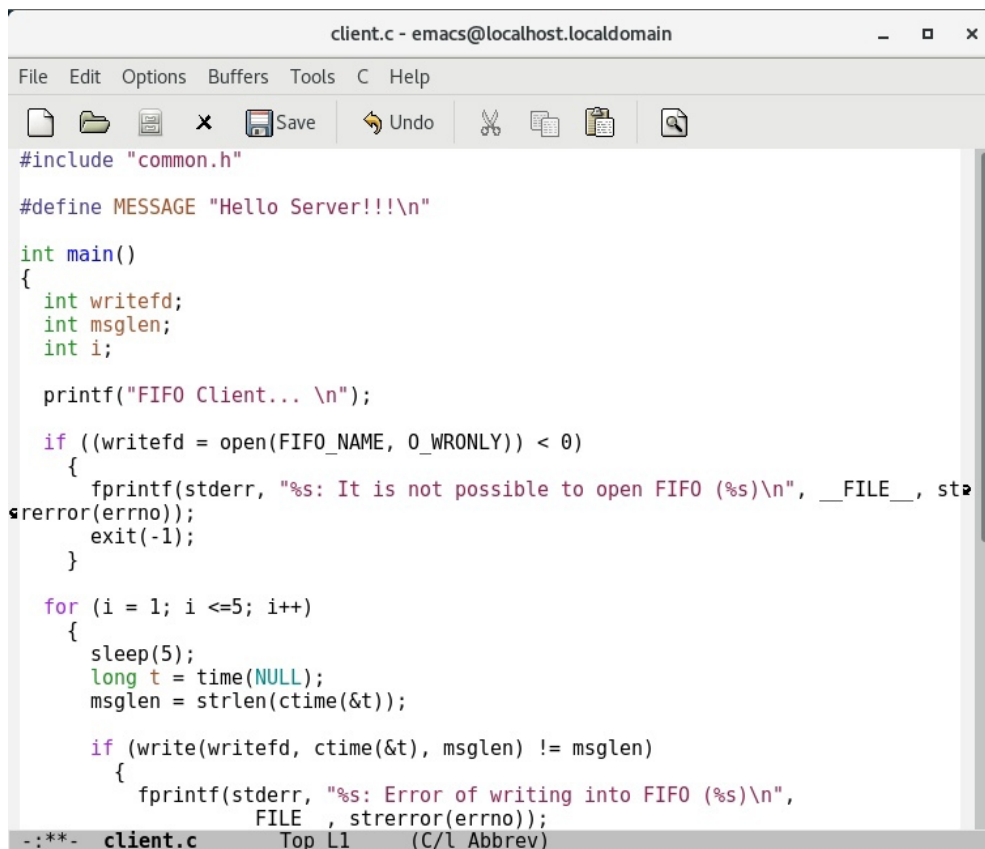
if(unlink(FIFO_NAME) < 0)
{
    fprintf(stderr, "%s: It is not possible to delete FIFO (%s)\n",
            __FILE__, strerror(errno));
    stop = clock();
    double time = (double)(stop - start) / CLOCKS_PER_SEC;
    printf("Time of the process: %f seconds\n", time);
    exit(-4);
}

stop = clock();
double time = (double)(stop - start) / CLOCKS_PER_SEC;
printf("Time of the process: %f seconds\n", time);

exit(0);
}

--:--- server.c      56% L56      (C/l Abbrev)
```

рисунок 5: файл server.c



```
client.c - emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
Save Undo
#include "common.h"
#define MESSAGE "Hello Server!!!\n"
int main()
{
    int writefd;
    int msglen;
    int i;

    printf("FIFO Client... \n");

    if ((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
    {
        fprintf(stderr, "%s: It is not possible to open FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-1);
    }

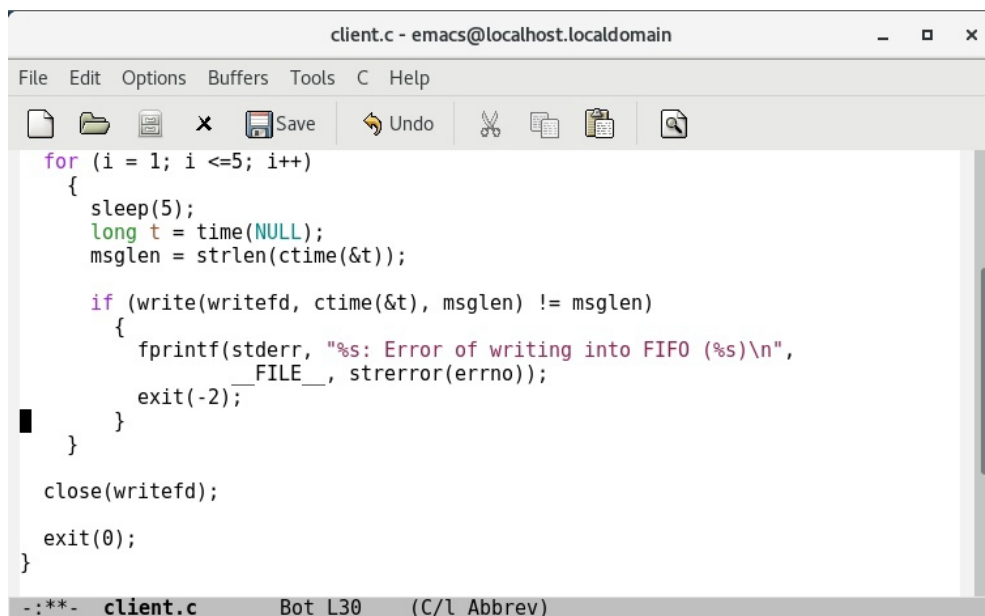
    for (i = 1; i <=5; i++)
    {
        sleep(5);
        long t = time(NULL);
        msglen = strlen(ctime(&t));

        if (write(writefd, ctime(&t), msglen) != msglen)
        {
            fprintf(stderr, "%s: Error of writing into FIFO (%s)\n", __FILE__, strerror(errno));
            exit(-1);
        }
    }

    close(writefd);
    exit(0);
}
```

client.c Top L1 (C/l Abbrev)

рисунок 6: файл client.c



```
client.c - emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
Save Undo
for (i = 1; i <=5; i++)
{
    sleep(5);
    long t = time(NULL);
    msglen = strlen(ctime(&t));

    if (write(writefd, ctime(&t), msglen) != msglen)
    {
        fprintf(stderr, "%s: Error of writing into FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-1);
    }
}

close(writefd);
exit(0);
}
```

client.c Bot L30 (C/l Abbrev)

рисунок 7: файл client.c

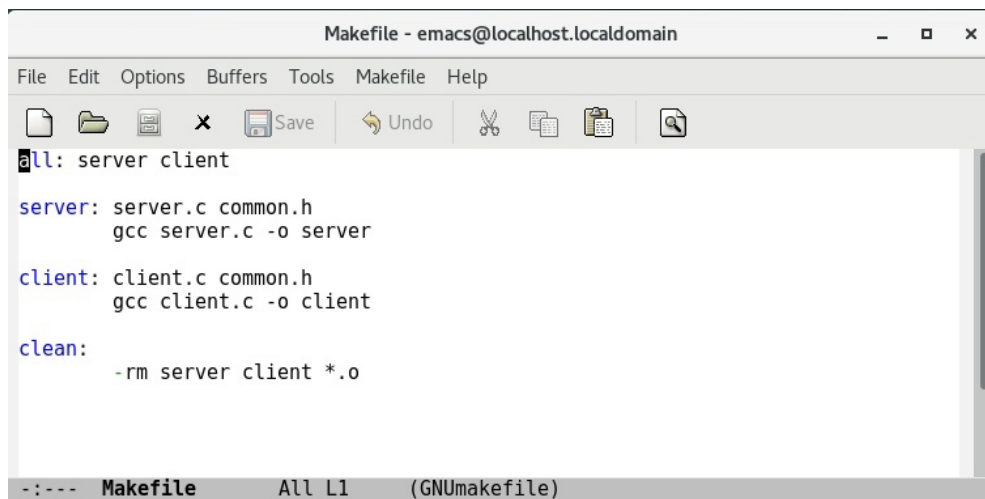


рисунок 8: файл Makefile

Какие коррективы были внесены в первоначальный код:

- в файле `server.c` появилась функция `clock()`, помогающая подсчитывать кол-во времени, затраченное на выполнение алгоритма, подробнее с данной функцией можно ознакомиться статье [Измерение времени выполнения блока кода на C/C++](#);
- в файле `client.c` вывод текущей даты и времени осуществляем 5 раз с интервалом в 5 секунд - `sleep(5)` ;

Далее компилируем наши программы `server.c` и `client.c` при помощи компилятора `gcc` (рисунок 7). Система не выдает нам ошибок, следовательно все реализовано верно.

2. Проверяем работу программ (рисунок 9). Откроем три терминала, в одном из них первую очередь запускаем `server.c`, а в оставшихся двух - два `./client.c`.

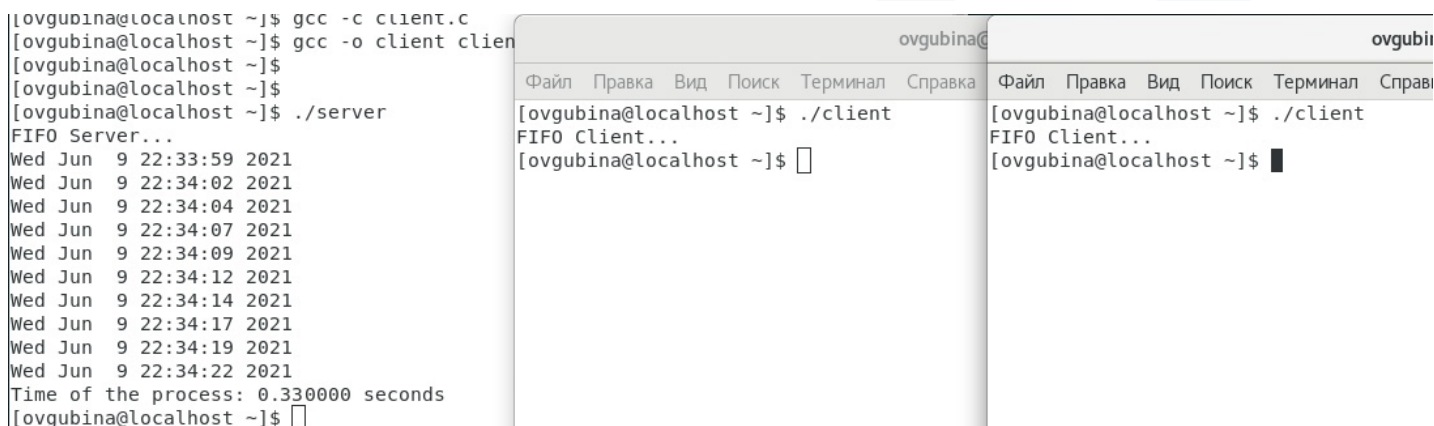


рисунок 9: работа программ

Видим, что оба клиента выводят дату и время, можно заметить что каждый из них делает это с интервалов в 5 секунд, а интервалы между выводами разных клиентов равно разнице во времени из запуска, в нашем случае - 3 секунды. Отсюда и итоговое время выполнения работы: 0.33 sec.

## Вывод:

Приобрела практические навыки работы с именованными каналами.

## Библиография:

[1]: [Каналы FIFO](#)

[2]: [Измерение времени выполнения блока кода на C/C++](#)

## Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Наличием идентификатора канала, который представлен как специальный файл.

2. Возможно ли создание неименованного канала из командной строки?

Это можно сделать при помощи команды `pipe`.

3. Возможно ли создание именованного канала из командной строки?

```
$ mkfifo [имя_файла]
```

4. Опишите функцию языка C, создающую неименованный канал.

```
int read(int pipe_fd, void *area, int cnt);
int write(int pipe_fd, void *area, int cnt);
```

Первый аргумент этих вызовов - дескриптор канала, второй - указатель на область памяти, с которой происходит обмен, третий - количество байт.

**5. Опишите функцию языка C, создающую именованный канал.**

```
int mkfifo (const char *pathname, mode_t mode)
```

Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`): `mkfifo(FIFO_NAME, 0600)`.

**6. Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов?**

При чтении меньшего числа байтов, чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO возвращается доступное число байтов.

**7. Аналогично, что будет в случае записи в `fifo` меньшего числа байтов, чем позволяет буфер? Большого числа байтов?**

При записи большего числа байтов, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал. Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются.

**8 Могут ли два и более процессов читать или записывать в канал?**

В общем случае возможна ситуация, когда с одним и тем же каналом взаимодействуют два и более процесса, и каждый из взаимодействующих каналов пишет и читает информацию в канал. Но традиционной схемой организации работы с каналом является однонаправленная организация, когда канал связывает два, в большинстве случаев, или несколько взаимодействующих процесса, каждый из которых может либо читать, либо писать в канал.

**9. Опишите функцию `write` (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе `server.c` (строка 42)?**

Функция записывает `length` байтов из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция реализуется как непосредственный вызов DOS. С помощью функции `write` мы посылаем сообщение клиенту или серверу.

**10. Опишите функцию `strerror`.**

Обычно хранится в глобальной переменной `errno`, в сообщение об ошибке, понятном человеку. Ошибки эти возникают при вызове функций стандартных Си-библиотек. Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции `strerror` перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.