

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 15

дисциплина: Операционные системы

Студент: Ким Реачна

Группа: НПИбд-02-20

Москва

2021г.

Цель работы:

Приобретение практических навыков работы с именованными каналами.

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: общеюниксные (именованные каналы, сигналы System V Interface Definition (SVID — разделяемая память, очередь сообщений семафоры) и 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.

После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения. При закрытии файла сам канал продолжает существовать. Для того чтобы закрыть сам канал, нужно удалить его файл, например с помощью вызова `unlink(2)`.

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

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

В качестве маски доступа используется восьмеричное значение `0600`, разрешающее процессу с аналогичными реквизитами пользователя чтение и запись. Можно также установить права доступа `0666`.

Открываем созданный файл для чтения:

```
f = fopen(FIFO_NAME, O_RDONLY);
```

Клиент открывает FIFO для записи как обычный файл:

```
f = fopen(FIFO_NAME, O_WRONLY);
```

Более подробная информация о лаборатории № 15. [1](#)

Задание:

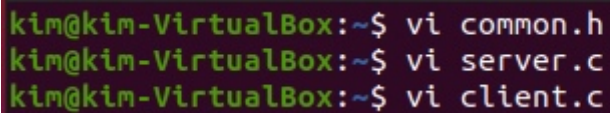
Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения:

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. Создайте файлы `server.c`, `client.c` `common.h` и `Makefile` с помощью редактора команд `vi` (Рисунок 1).

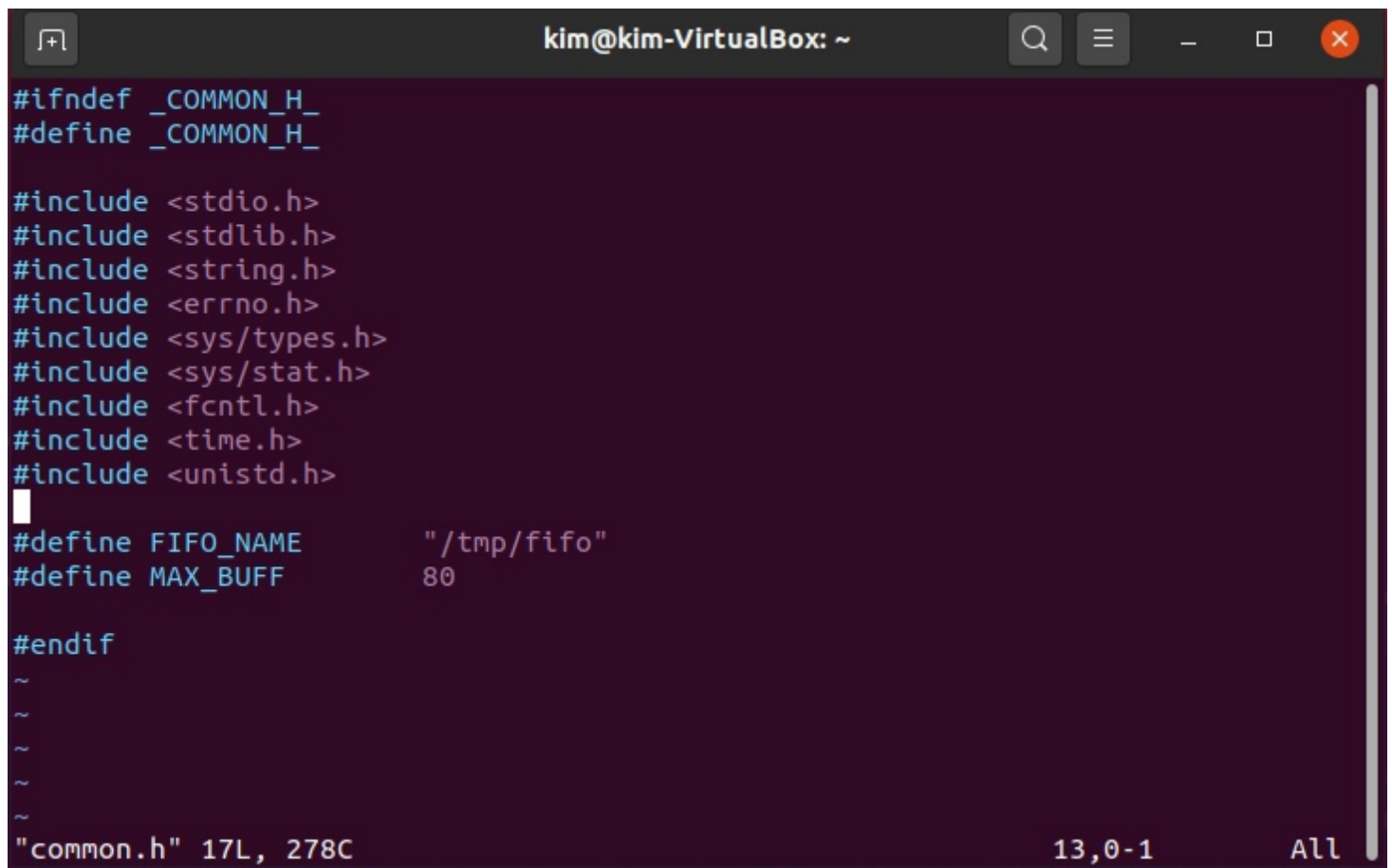
Рисунок 1: Создайте файлы



```
kim@kim-VirtualBox:~$ vi common.h
kim@kim-VirtualBox:~$ vi server.c
kim@kim-VirtualBox:~$ vi client.c
```

Рисунок 2: Файл `common.h`

В этом общем файле.h я добавил `#include <time.h>` и `#include <unistd.h>`, чтобы мы могли запустить работу командного файла.(Рисунок 2)



```
#ifndef _COMMON_H_
#define _COMMON_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <unistd.h>

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

#endif
~
~
~
~
~
"common.h" 17L, 278C                                13,0-1                                All
```

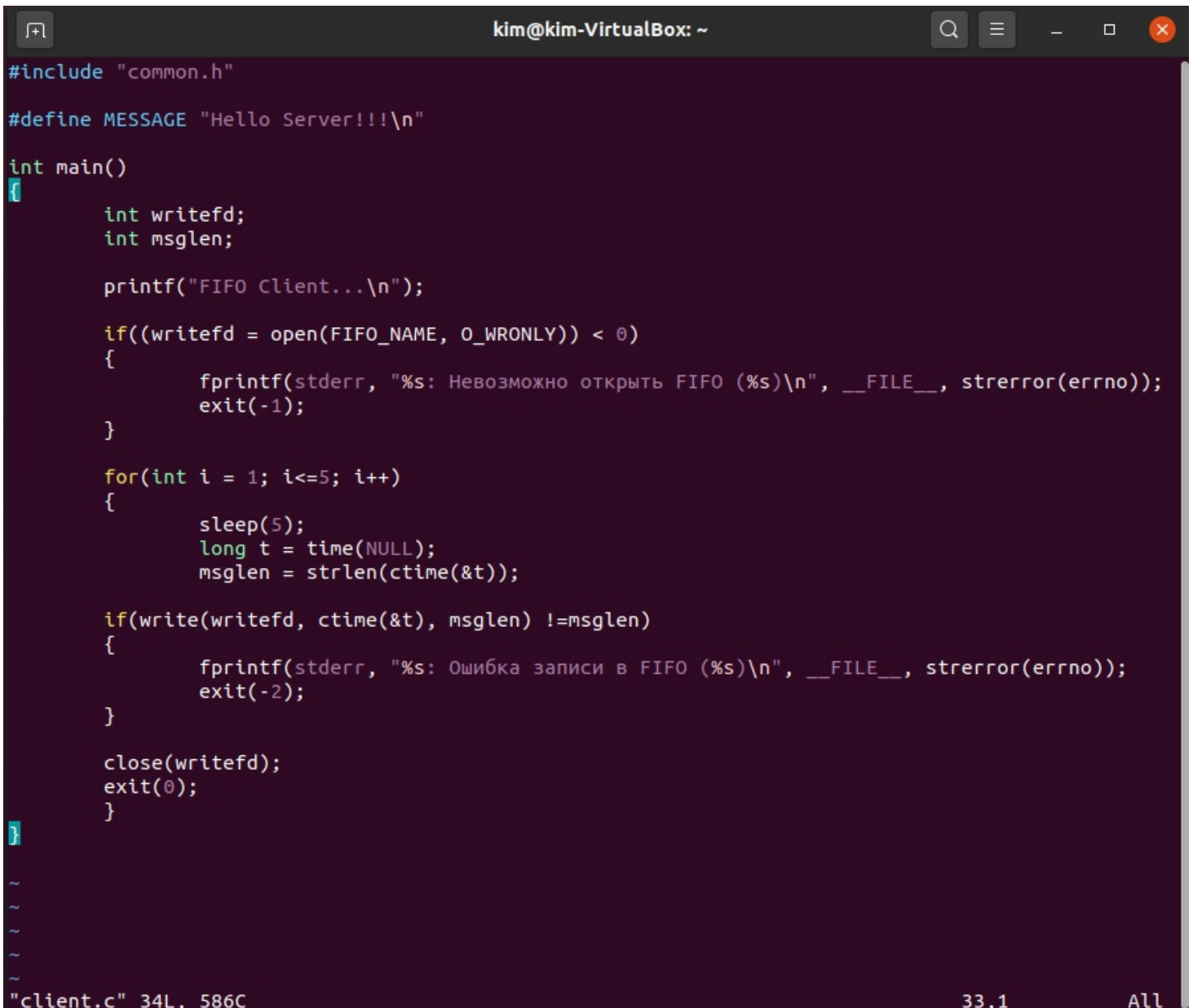
Рисунок 3: Файл `server.c`

```
kim@kim-VirtualBox: ~  
#include "common.h"  
  
int main()  
{  
    int readfd;  
    int n;  
    clock_t startpoint, stoppoint;  
    char buff[MAX_BUFF];  
  
    printf("FIFO Server...\n");  
  
    startpoint = clock();  
  
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)  
    {  
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n", __FILE__, strerror(errno));  
        stoppoint = clock();  
        double time = (double)(stoppoint - startpoint) / 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: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));  
        stoppoint = clock();  
        double time = (double)(stoppoint - startpoint) / CLOCKS_PER_SEC;  
        printf("Time of the process: %f seconds\n", time);  
        exit(-2);  
    }  
}
```

Рисунок 4: Файл server.c

```
while((n = read (readfd, buff, MAX_BUFF)) > 0)  
{  
    if(write(1, buff, n) !=n)  
    {  
        fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));  
        stoppoint = clock();  
        double time = (double)(stoppoint - startpoint) / CLOCKS_PER_SEC;  
        printf("Time of the process: %f seconds\n", time);  
        exit(-3);  
    }  
}  
  
close(readfd);  
  
if(unlink (FIFO_NAME) < 0)  
{  
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));  
    stoppoint = clock();  
    double time = (double)(stoppoint - startpoint) / CLOCKS_PER_SEC;  
    printf("Time of the process: %f seconds\n", time);  
    exit(-4);  
}  
  
exit(0);  
}
```

Рисунок 5: Файл client.c



```
#include "common.h"

#define MESSAGE "Hello Server!!!\n"

int main()
{
    int writefd;
    int msglen;

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

    if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-1);
    }

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

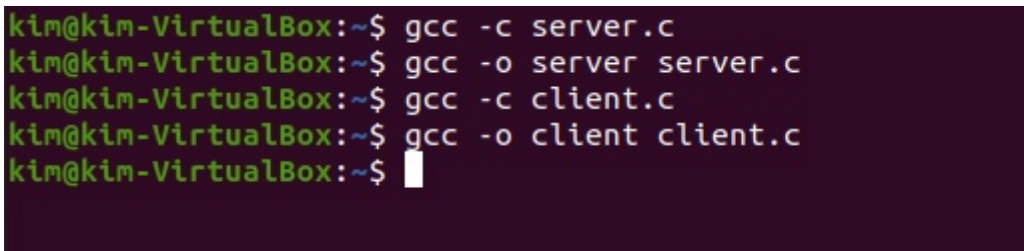
        if(write(writefd, ctime(&t), msglen) !=msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__, strerror(errno));
            exit(-2);
        }

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

"client.c" 34L, 586C 33,1 All

Рисунок 6: Проверка файлов

Используя команды `gcc -c server.c` и `gcc -c client.c`, чтобы проверить, есть ли какие-либо ошибки в наших командных файлах (Рисунок 6).



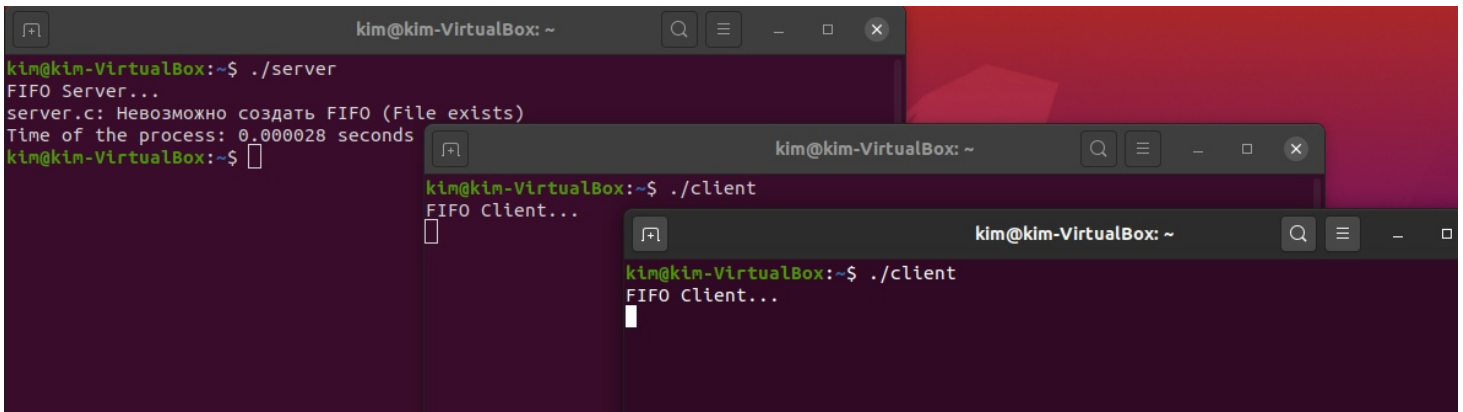
```
kim@kim-VirtualBox:~$ gcc -c server.c
kim@kim-VirtualBox:~$ gcc -o server server.c
kim@kim-VirtualBox:~$ gcc -c client.c
kim@kim-VirtualBox:~$ gcc -o client client.c
kim@kim-VirtualBox:~$
```

Как мы видим, в наших командных файлах нет ошибок.

2. Проверьте нашу работу: после проверки того, что в наших командных файлах нет ошибок, теперь мы проверим нашу целевую работу с помощью команд `./server` и `./client`

(Рисунок 7).

Рисунок 7: Работа над файлом



The image shows three overlapping terminal windows from a VirtualBox environment. The top-left window shows the execution of a server program: `kim@kim-VirtualBox:~$./server`, followed by `FIFO Server...`, an error message `server.c: Невозможно создать FIFO (File exists)`, and the process time `Time of the process: 0.000028 seconds`. The top-right window shows the execution of a client program: `kim@kim-VirtualBox:~$./client`, followed by `FIFO Client...`. The bottom window also shows the execution of a client program: `kim@kim-VirtualBox:~$./client`, followed by `FIFO Client...`. The background of the VirtualBox window is a red and black gradient.

Вывод:

Я познакомилась с практическими навыками работы с названными каналами.

Библиография:

[1]:[Материал лаборатории 15](#)

[2]:[Последствия, объявленные функциями в нашем файле command.h](#)

[3]:[Именованный канал](#)

Контрольные вопросы:

1. Именованные каналы, в отличие от неименованных, могут использоваться неродственными процессами. Они дают вам, по сути, те же возможности, что и неименованные каналы, но с некоторыми преимуществами, присущими обычным файлам.
2. Да, для создания неименованного канала используется системный вызов `pipe`.
3. Да, `$ 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);
```

5. Опишите функцию языка C, создающую именованный канал.

```
int mkfifo (const char *pathname, mode_t_mode)
```

6. При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем заказано.
7. При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений.
8. орожденные процессы-братья: родительский вызывает `pipe` для создания канала, затем порождает два или больше процессов-братьев. Порожденные процессы могут общаться по каналу посредством своих дескрипторов `fifo[0]` и `fifo[1]`.
9. 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. Функция `strerror()` возвращает строку, описывающую код ошибки, переданный в аргументе `errnum`, возможно с учетом категории `LC_MESSAGES` текущей локали для выбора соответствующего языка.