

# **Отчет по лабораторной работе №14**

**Дисциплина: операционные системы**

Лобанова Полина Иннокентьевна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	10
4	Вывод	13

## Список иллюстраций

2.1	Создание каталога и файлов. . . . .	6
2.2	Содержание файла <i>common.h</i> . . . . .	7
2.3	Содержание файла <i>server.c</i> . . . . .	7
2.4	Содержание файла <i>client.c</i> . . . . .	8
2.5	Содержание файла <i>client2.c</i> . . . . .	8
2.6	Содержание файла <i>Makefile</i> . . . . .	9
2.7	Компиляция. . . . .	9
2.8	Результат программы. . . . .	9

## Список таблиц

# **1 Цель работы**

Приобретение практических навыков работы с именованными каналами.

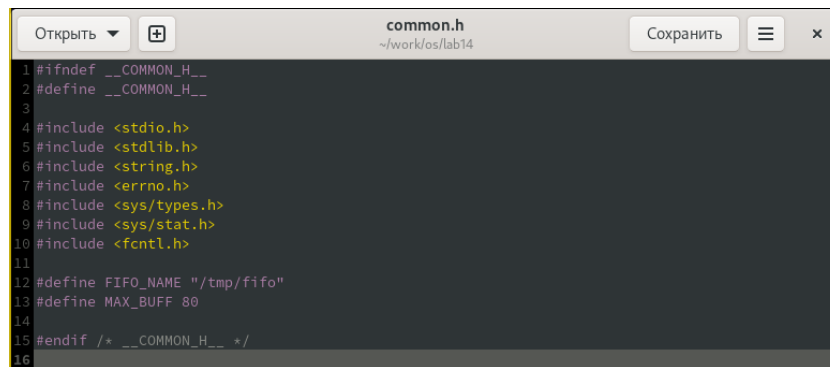
## 2 Выполнение лабораторной работы

1. Создадим необходимые каталог и файлы.

```
[pilobanova@fedora ~]$ cd ~/work/os/  
[pilobanova@fedora os]$ mkdir lab14  
[pilobanova@fedora os]$ cd lab14  
[pilobanova@fedora lab14]$ touch common.h  
[pilobanova@fedora lab14]$ gedit common.h  
[pilobanova@fedora lab14]$ touch server.c  
[pilobanova@fedora lab14]$ touch client.c  
[pilobanova@fedora lab14]$ touch client2.c  
[pilobanova@fedora lab14]$ touch Makefile
```

Рис. 2.1: Создание каталога и файлов.

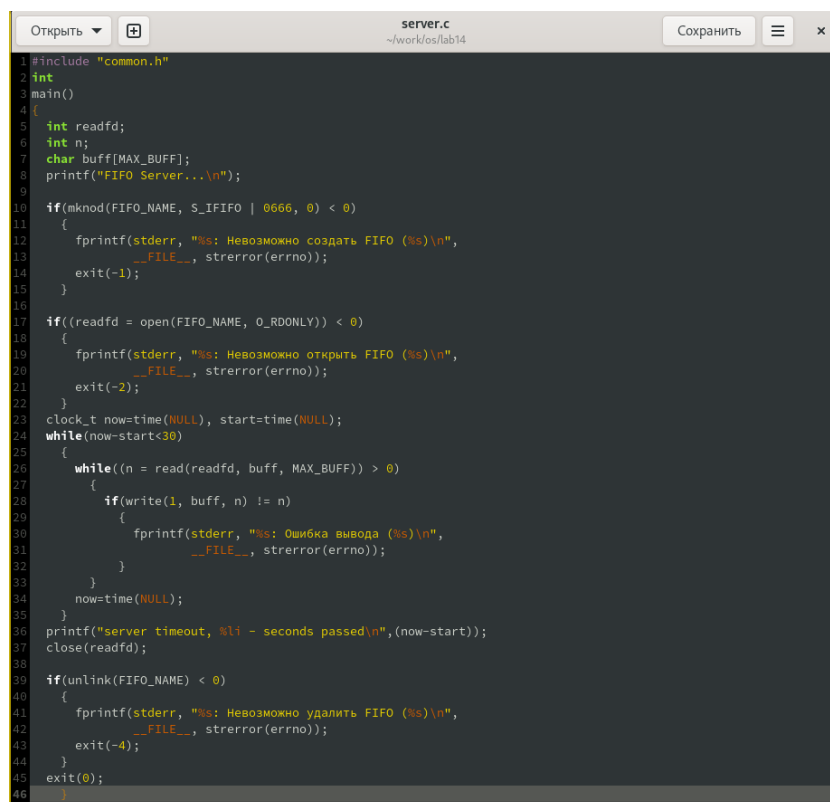
2. Изучим приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напомним аналогичные программы, внеся следующие изменения:
  - 1) Работает не 1 клиент, а несколько (например, два).
  - 2) Клиенты передают текущее время с некоторой периодичностью (например, раз в пять 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.



The screenshot shows a code editor window titled "common.h" with the path "~/work/os/lab14". The editor contains the following C code:

```
1 #ifndef __COMMON_H__
2 #define __COMMON_H__
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <sys/types.h>
9 #include <sys/stat.h>
10 #include <fcntl.h>
11
12 #define FIFO_NAME "/tmp/fifo"
13 #define MAX_BUFF 80
14
15 #endif /* __COMMON_H__ */
16
```

Рис. 2.2: Содержание файла *common.h*.



The screenshot shows a code editor window titled "server.c" with the path "~/work/os/lab14". The editor contains the following C code:

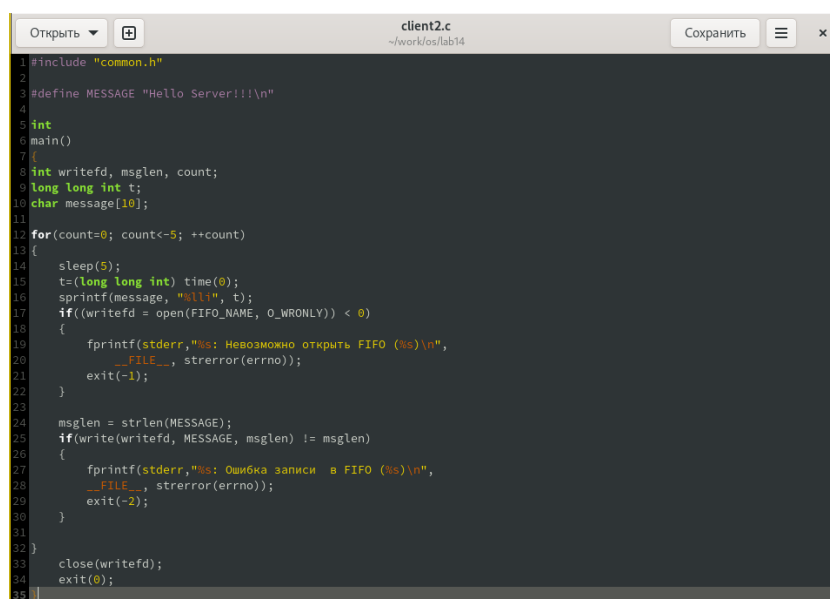
```
1 #include "common.h"
2 int
3 main()
4 {
5     int readfd;
6     int n;
7     char buff[MAX_BUFF];
8     printf("FIFO Server...\n");
9
10    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
11    {
12        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
13            __FILE__, strerror(errno));
14        exit(-1);
15    }
16
17    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
18    {
19        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20            __FILE__, strerror(errno));
21        exit(-2);
22    }
23    clock_t now=time(NULL), start=time(NULL);
24    while(now-start<30)
25    {
26        while((n = read(readfd, buff, MAX_BUFF)) > 0)
27        {
28            if(write(1, buff, n) != n)
29            {
30                fprintf(stderr, "%s: Ошибка вывода (%s)\n",
31                    __FILE__, strerror(errno));
32            }
33        }
34        now=time(NULL);
35    }
36    printf("server timeout, %li ~ seconds passed\n", (now-start));
37    close(readfd);
38
39    if(unlink(FIFO_NAME) < 0)
40    {
41        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
42            __FILE__, strerror(errno));
43        exit(-4);
44    }
45    exit(0);
46 }
```

Рис. 2.3: Содержание файла *server.c*.



```
1 #include "common.h"
2
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int msg, len, i;
9     long int t;
10
11     for(i=0; i<20; i++)
12     {
13         sleep(3);
14         t=time(NULL);
15         printf("FIFO Client...\n");
16
17         if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
18         {
19             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20                     __FILE__, strerror(errno));
21             exit(-1);
22         }
23
24         len = strlen(MESSAGE);
25
26         if(write(msg, MESSAGE, len) != len)
27         {
28             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
29                     __FILE__, strerror(errno));
30             exit(-2);
31         }
32         close(msg);
33     }
34     exit(0);
35 }
```

Рис. 2.4: Содержание файла *client.c*.



```
1 #include "common.h"
2
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int writefd, msglen, count;
9     long long int t;
10     char message[10];
11
12     for(count=0; count<5; ++count)
13     {
14         sleep(5);
15         t=(long long int) time(0);
16         sprintf(message, "%lli", t);
17         if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
18         {
19             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20                     __FILE__, strerror(errno));
21             exit(-1);
22         }
23
24         msglen = strlen(MESSAGE);
25         if(write(writefd, MESSAGE, msglen) != msglen)
26         {
27             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
28                     __FILE__, strerror(errno));
29             exit(-2);
30         }
31
32     }
33     close(writefd);
34     exit(0);
35 }
```

Рис. 2.5: Содержание файла *client2.c*.



```
Makefile
~/work/os/lab14

1 all: server client
2
3 server: server.c common.h
4     gcc server.c -o server
5
6 client: client.c common.h
7     gcc client.c -o client
8
9 clean:
10    -rm server client *.o
```

Рис. 2.6: Содержание файла Makefile.

### 3. Скомпилируем файл.

```
pi@pobanova@fedora lab14$ make
gcc server.c -o server
server.c:8: функция «main»:
server.c:21:18: предупреждение: неиспользуемая переменная «time» [-Wunused-variable]
21 |     clock_t now=time(NULL), start=time(NULL);
    |                   ^~~~~
server.c:21:1: warning: «time» is defined in header «<time.h>»; did you forget to «#include <time.h>»?
1 | #include <time.h>
2 | int
3 | int
server.c:21:18: предупреждение: неиспользуемая переменная «read»; имеется в виду «fread» [-Wunused-variable]
26 |     while(n < read(readfd, buff, MAX_BUF)) < 0)
    |                   ^~~~~
server.c:21:18: предупреждение: неиспользуемая переменная «write»; имеется в виду «fwrite» [-Wunused-variable]
28 |     if(write(1, buff, n) < 0)
    |                   ^~~~~
server.c:27:1: предупреждение: неиспользуемая переменная «close»; имеется в виду «fclose» [-Wunused-variable]
27 |     close(readfd);
    |     ^~~~~
server.c:30:18: предупреждение: неиспользуемая переменная «unlink» [-Wunused-variable]
30 |     if(unlink(FIFO_NAME) < 0)
    |                   ^~~~~
gcc client.c -o client
client.c:8: функция «main»:
client.c:11:18: предупреждение: неиспользуемая переменная «sleep» [-Wunused-variable]
12 |     sleep(1);
    |                   ^~~~~
client.c:14:1: предупреждение: неиспользуемая переменная «time»; имеется в виду «fopen» [-Wunused-variable]
14 |     time(NULL);
    |     ^~~~~
client.c:21:1: warning: «time» is defined in header «<time.h>»; did you forget to «#include <time.h>»?
1 | #include <time.h>
2 | int
3 | int
client.c:21:18: предупреждение: неиспользуемая переменная «write»; имеется в виду «fwrite» [-Wunused-variable]
26 |     if(write(writefd, msg, 1) < 0)
    |                   ^~~~~
client.c:32:1: предупреждение: неиспользуемая переменная «close»; имеется в виду «fclose» [-Wunused-variable]
32 |     close(writefd);
    |     ^~~~~
pi@pobanova@fedora lab14$
```

Рис. 2.7: Компиляция.

### 4. Запустим программу.

```
pi@pobanova@fedora lab14$ ./server
FIFO Server...
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
Hello Server!!
server timeout, 30 - seconds passed

pi@pobanova@fedora lab14$ ./client
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
client.c: Неудачно открыть FIFO (No such file or directory)
```

Рис. 2.8: Результат программы.

### 3 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла).

2. Возможно ли создание неименованного канала из командной строки?

Создание неименованного канала из командной строки возможно командой `pipe`.

3. Возможно ли создание именованного канала из командной строки?

Создание именованного канала из командной строки возможно с помощью `mkfifo`.

4. Опишите функцию языка C, создающую неименованный канал.

Функция языка C, создающая 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. Оба вызова возвращают число переданных байт (или -1 - при ошибке).

5. Опишите функцию языка C, создающую именованный канал.

Функция языка C, создающая именованный канал: `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);`

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

При чтении меньшего числа байтов, возвращается требуемое число байтов, остаток сохраняется для следующих чтений. При чтении большего числа байтов, возвращается доступное число байтов.

7. 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. 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 `SIGPIPE`, а вызов `write(2)` возвращает 0 с установкой ошибки (`errno=EP1PE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

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

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. 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 SIGPIPE, а вызов `write(2)` возвращает 0 с установкой ошибки (`errno=EP1PE`) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию – процесс завершается).

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

Два и более процессов могут читать и записывать 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 `write` записывает `length` байтов из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция чисто ‘двоичная’ и без буферизации. При единице возвращает действительное число байтов. Функция `write` возвращает число действительно записанных в файл байтов или -1 при ошибке, устанавливая при этом `errno`.

10. Опишите функцию `strerror`

Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной `errno`, в сообщение об ошибке, понятном человеку.

## **4 Вывод**

Я приобрела практические навыки работы с именованными файлами.