

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

### Кафедра прикладной информатики и теории вероятностей

#### ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 15 «Именованные каналы»

дисциплина: *Операционные системы*

Студент: Юсупов Шухратджон Фирдавсович

Группа: НПИбд 02-20

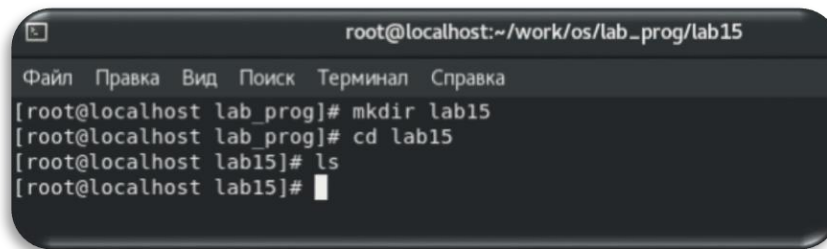
МОСКВА 2021 г.

---

**Цель работы:** Приобретение практических навыков работы с именованными каналами.

#### Ход работы

1. Мы создали нужные файлы и сделали их исполняемыми. (Рис. 1)



```
root@localhost:~/work/os/lab_prog/lab15
Файл Правка Вид Поиск Терминал Справка
[root@localhost lab_prog]# mkdir lab15
[root@localhost lab_prog]# cd lab15
[root@localhost lab15]# ls
[root@localhost lab15]#
```

рис. 1

2. Изучим приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напомним аналогичные программы, внеся следующие изменения:
  - Работает не 1 клиент, а несколько (например, два).
  - Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд).
  - Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек).

server.c (Рис. 2-3)

```
#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 isn't 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)
    {
```

рис. 2

```
    while ((n = read(readfd, buff, MAX_BUFF)) > 0)
    {
        if(write(1, buff, n) != n)
        {
            fprintf(stderr, "%s: Error (%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 isn't 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);
}
```

рис. 3

*client.c* (Рис. 4-5)

```
#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 isn't 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(-2);
        }
    }
}
```

рис. 4

```
    {
        fprintf(stderr, "%s: Error of writing into FIFO (%s)\n",
                __FILE__, strerror(errno));
        exit(-2);
    }
}

close(writefd);

exit(0);

}
```

) рис. 5

```

#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>

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

#endif

```

common.h (Рис. 6)

рис. 6

```

all: server client

server: server.c common.h
gcc server.c -o server
client: client.c common.h
gcc client.c -o client
clean:
-rm server client *.o

```

Makefile (Рис. 7)

рис. 7

## ВЫВОД:

В ходе работы я приобрел практические навыки работы с именованными каналами

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Воспользоваться интернетом, воспользоваться командой man, info
2. Unix поддерживает следующие основные этапы разработки приложений:
  - создание исходного кода программы;
  - сохранение различных вариантов исходного текста;
  - анализ исходного текста; Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.
  - компиляция исходного текста и построение исполняемого модуля
  - тестирование и отладка;

-сохранение всех изменений, выполняемых при тестировании и отладке.

3. Использование суффикса ".c" для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c.
4. В компиляции всей программы в целом и получении исполняемого модуля.
5. Для упрощения и автоматизации работы пользователя с командной строкой
6. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат:

```
target1 [ target2...]: [:] [dependment1...]
```

```
[(tab)commands]
```

```
[#commentary]
```

```
[(tab)commands]
```

```
[#commentary],
```

где # — специфицирует начало комментария; : — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний.

7. Все программы отладки позволяют отслеживать состояние программы на любом из этапов ее исполнения. Для того чтобы эту возможность использовать необходимо изучить документацию по использованию определенного отладчика. Понять общие принципы отладки.
8. – backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;  
– break – устанавливает точку останова; параметром может быть номер строки или название функции;  
– clear – удаляет все точки останова на текущем уровне стека (то есть

в текущей функции);

- continue – продолжает выполнение программы от текущей точки до конца;
- delete – удаляет точку останова или контрольное выражение;
- display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
- finish – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
- info breakpoints – выводит список всех имеющихся точек останова;
- info watchpoints – выводит список всех имеющихся контрольных выражений;
- list – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
- next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;
- print – выводит значение какого-либо выражения (выражение передается в качестве параметра);
- run – запускает программу на выполнение;
- set – устанавливает новое значение переменной
- step – пошаговое выполнение программы;
- watch – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;

9. Сначала запустил отладчик для программы. Установил интересующую меня точку останова. Запустил программу ожидая, что программа остановится на точке останова. Узнал необходимые данные моей программы на текущем этапе ее исполнения путем ввода команд. Отобразил данные. Завершил программу, снял точки останова.
10. В моем случае не было синтаксических ошибок. Были ошибки семантические. Компилятор начал жаловаться на то, что программа по смыслу принимает указатель на char массив. В то время как я вводил не указатель, а прямое значение. Ошибка была исправлена путем удаления &.

- cscope - исследование функций, содержащихся в программе;
- lint -- критическая проверка программ, написанных на языке Си.

11. Splint- инструмент для статической проверки С-программ на наличие уязвимостей и ошибок