

# Front matter

lang: ru-RU title: "отчёта по лабораторной работе 15" subtitle: "Операционные системы" author: "Алхатиб Осама"

## Formatting

toc-title: "Содержание" toc: true # Table of contents toc\_depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4paper documentclass: scrreprt polyglossia-lang: russian polyglossia-otherlangs: english mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions: Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase indent: true pdf-engine: lualatex header-includes: - \linepenalty=10 # the penalty added to the badness of each line within a paragraph (no associated penalty node) Increasing the value makes tex try to have fewer lines in the paragraph. - \interlinepenalty=0 # value of the penalty (node) added after each line of a paragraph. - \hyphenpenalty=50 # the penalty for line breaking at an automatically inserted hyphen - \exhyphenpenalty=50 # the penalty for line breaking at an explicit hyphen - \binoppenalty=700 # the penalty for breaking a line at a binary operator - \relpenalty=500 # the penalty for breaking a line at a relation - \clubpenalty=150 # extra penalty for breaking after first line of a paragraph - \widowpenalty=150 # extra penalty for breaking before last line of a paragraph - \displaywidowpenalty=50 # extra penalty for breaking before last line before a display math - \brokenpenalty=100 # extra penalty for page breaking after a hyphenated line - \predisplaypenalty=10000 # penalty for breaking before a display - \postdisplaypenalty=0 # penalty for breaking after a display - \floatingpenalty = 20000 # penalty for splitting an insertion (can only be split footnote in standard LaTeX) - \raggedbottom # or \flushbottom - \usepackage{float} # keep figures where there are in the text

- \floatplacement{figure}{H} # keep figures where there are in the text

## Отчет по лабораторной работе №15

Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

### Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Бен бадр Ясмин

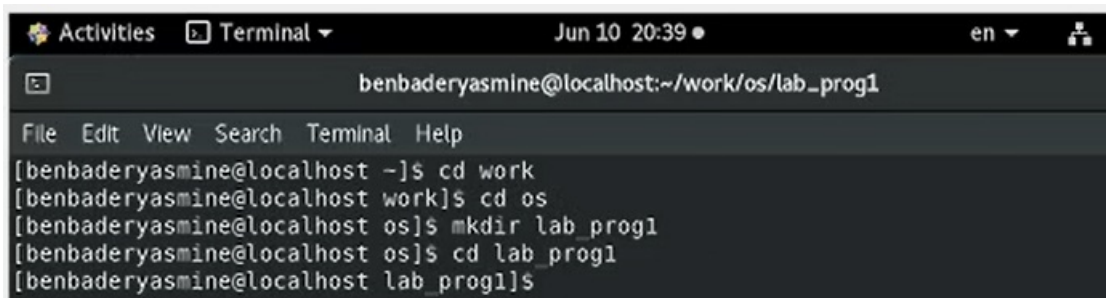
Группа: НКНДб-01-20

2021г.

### Цель работы

Приобретение практических навыков работы с именованными каналами.

Ход работы 1. Перешла в рабочий каталогах `work/os/lab_prog1`, чтобы начинать сделать лаб15



```
benbaderyasmine@localhost:~/work/os/lab_prog1
File Edit View Search Terminal Help
[benbaderyasmine@localhost ~]$ cd work
[benbaderyasmine@localhost work]$ cd os
[benbaderyasmine@localhost os]$ mkdir lab_prog1
[benbaderyasmine@localhost os]$ cd lab_prog1
[benbaderyasmine@localhost lab_prog1]$
```

2. создала три файла и в каждом написала программу

```
[benbaderyasmine@localhost lab_prog1]$ touch common.h server.c client.c makefile
[benbaderyasmine@localhost lab_prog1]$ emacs
```

Файл common.h

```
/*
 * common.h - заголовочный файл со стандартными определениями
 */

#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 */

U:--- common.h    All L1    (C/*\ Abbrev)
```

osyasmine1 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

```
Activities Emacs Jun 10 20:45 en
emacs@localhost.localdomain

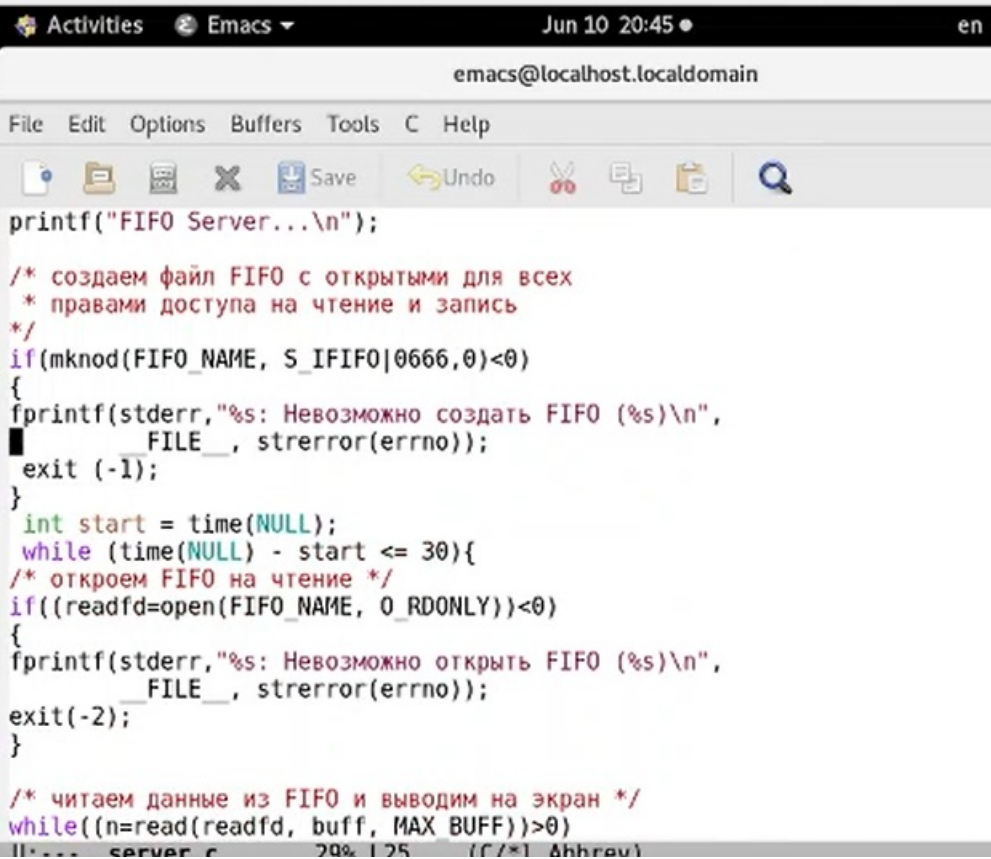
File Edit Options Buffers Tools C Help

/*
 * server.c - реализация сервера
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"

int
main()
{
    int readfd; /* дескриптор для чтения из FIFO */
    int n;
    char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */
    /* баннер */
    printf("FIFO Server...\n");

    /* создаем файл FIFO с открытыми для всех
     * правами доступа на чтение и запись
     */
    if(mknod(FIFO_NAME, S_IFIFO|0666,0)<0)
    {
U:--- server.c    Top L1    (C/*\ Abbrev)
```

Файл server.c



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

/* создаем файл FIFO с открытыми для всех
 * правами доступа на чтение и запись
 */
if(mknod(FIFO_NAME, S_IFIFO|0666,0)<0)
{
fprintf(stderr,"%s: Невозможно создать FIFO (%s)\n",
__FILE__, strerror(errno));
exit(-1);
}
int start = time(NULL);
while (time(NULL) - start <= 30){
/* откроем FIFO на чтение */
if((readfd=open(FIFO_NAME, O_RDONLY))<0)
{
fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
__FILE__, strerror(errno));
exit(-2);
}

/* читаем данные из FIFO и выводим на экран */
while((n=read(readfd, buff, MAX_BUFF))>0)
ll: ... server.c 20% 1.25 (C/*1 Abbrev)
```

```
__FILE__, strerror(errno));
exit(-2);
}
/* читаем данные из FIFO и выводим на экран */
while((n=read(readfd, buff, MAX_BUFF))>0)
{
    if(write(1, buff, n)!=n)
    {
        fprintf(stderr,"%s: Ошибка вывода (%s)\n",
            __FILE__, strerror(errno));
        exit(-3);
    }
}
close(readfd);/* закроем FIFO */

/* удалим FIFO из системы */
if(unlink(FIFO_NAME)<0)
{
    fprintf(stderr,"%s: Невозможно удалить FIFO (%s)\n",
        __FILE__, strerror(errno));
    exit(-4);
}
```

```
/*
 * client.c - реализация клиента
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
```

```
#include "common.h"
```

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

```
int
```

```
main()
```

```
{
```

```
int writefd; /* дескриптор для записи в FIFO */
```

```
int msglen;
```

```
/* баннер */
```

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

```
/* получим доступ к FIFO */
```

```
if((writefd=open(FIFO_NAME, O_WRONLY))<0)
```

```
{
```

```
U:--- client.c Top L12 (C/*l Abbrev)
```

```

Activities Emacs Jun 10 20:46 en
emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
Save Undo
fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
        FILE_, strerror(errno));
exit(-1);
}

int i;
for (i=0; i<7; i++){
    sleep(7);
    long ttime = time(NULL);
    msglen = strlen(ctime(&ttime));

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

/* закроем доступ к FIFO */
close(writefd);

exit(0);
U:--- client.c 52% L26 (C/*l Abbrev)

```

```

Save Undo
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

U:***- makefile All L1 (GNUmakefile)

```

Файл makefile

3. разработалась в программах - примерах Сервер выключается после 3 сообщений в программе-примере сообщения принимались только определенной длины равной



```
benbaderyasmine@localhost:~/work/os/lab_prog1
File Edit View Search Terminal Help
[benbaderyasmine@localhost lab_prog1]$ make
gcc server.c -o server
server.c: In function 'main':
server.c:39:10: warning: implicit declaration of function 'read'; did you mean 'fread'
[-Wimplicit-function-declaration]
while((n=read(readfd, buff, MAX_BUFF))>0)
           ^
           fread
server.c:41:4: warning: implicit declaration of function 'write'; did you mean 'fwrite'
[-Wimplicit-function-declaration]
if(write(1, buff, n)!=n)
   ^
   fwrite
server.c:48:1: warning: implicit declaration of function 'close'; did you mean 'pclose'
[-Wimplicit-function-declaration]
close(readfd);/* закроем FIFO */
^
^
pclose
server.c:51:4: warning: implicit declaration of function 'unlink'; did you mean 'unix'
[-Wimplicit-function-declaration]
if(unlink(FIFO_NAME)<0)
   ^
   unix
gcc client.c -o client
client.c: In function 'main':
client.c:31:4: warning: implicit declaration of function 'sleep'; did you mean 'strse'
[-Wimplicit-function-declaration]
```

```
benbaderyasmine@localhost:~/work/os/lab_prog1
File Edit View Search Terminal Help
[benbaderyasmine@localhost lab_prog1]$ ./server
FIFO Server...
```

```
[benbaderyasmine@localhost lab_prog1]$ ./client
FIFO Client...
```

```
benbaderyasmine@localhost:~/work/os/lab_prog1$ ./client
FIFO Client...
```

```
[benbaderyasmine@localhost lab_prog1]$ ./server
FIFO Server...
Thu Jun 10 20:47:35 2021
Thu Jun 10 20:47:42 2021
Thu Jun 10 20:47:45 2021
Thu Jun 10 20:47:49 2021
Thu Jun 10 20:47:52 2021
Thu Jun 10 20:47:56 2021
Thu Jun 10 20:47:59 2021
Thu Jun 10 20:48:03 2021
Thu Jun 10 20:48:06 2021
Thu Jun 10 20:48:10 2021
Thu Jun 10 20:48:13 2021
Thu Jun 10 20:48:17 2021
Thu Jun 10 20:48:20 2021
Thu Jun 10 20:48:27 2021
[benbaderyasmine@localhost lab_prog1]$
```

4. получила результат

# Вывод

---

В результате работы , я приобрел практические навыки работы с именованными каналами

---

## Контрольные вопросы

1. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла).
2. Для создания неименованного канала используется системный вызов `pipe`. Массив из двух целых чисел является выходным параметром этого системного вызова.
3. Вы можете создавать именованные каналы из командной строки и внутри программы. С давних времен программой создания их в командной строке была команда: `mknod - $ mknod имяфайла` , однако команды `mknod` нет в списке команд *X/Open*, поэтому она включена не во все *UNIX-подобные системы*. Предпочтительнее применять в командной строке - `$ mkfifo имяфайла`.
4. `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. `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. 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. 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. В общем случае возможна многонаправленная работа процессов с каналом, т.е. 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. `Write` - Функция записывает `length` байтов из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция чисто 'двоичная' и без буферизации. Реализуется как непосредственный вызов `DOS`. С помощью функции `write` мы посылаем сообщение клиенту или серверу.
10. Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который 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