

Отчёт по лабораторной работе №15

Дисциплина: Операционные системы

Матвеева Анастасия Сергеевна

Содержание

1	Цель работы	4
2	Задачи	5
3	Выполнение лабораторной работы	6
4	Выводы	13
5	Контрольные вопросы	14
6	Библиография	18

List of Figures

3.1	Создание файлов и открытие редактора	6
3.2	Измененный файл common.h	7
3.3	Измененный файл server.c	8
3.4	Измененный файл server.c	9
3.5	Измененный файл client.c	10
3.6	Измененный файл client.c	10
3.7	Makefile	11
3.8	Компиляция	11
3.9	Проверка работы программы	11
3.10	Проверка работы программы	12

1 Цель работы

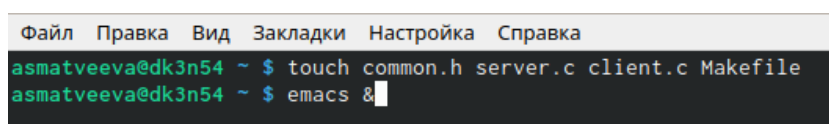
Приобретение практических навыков работы с именованными каналами.

2 Задачи

1. Познакомиться с механизмом именованных каналов и принципом FIFO.
2. Изучить данные в задании файлы.
3. В ходе работы изменить программы таким образом, чтобы выполнялись требуемые условия.

3 Выполнение лабораторной работы

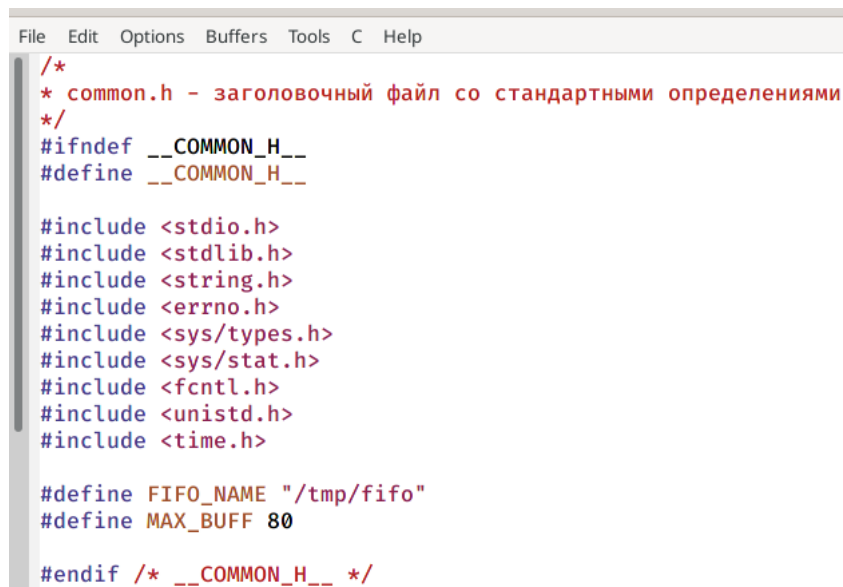
1. Для начала я создала необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» и открыла редактор emacs для их редактирования (рис. 3.1)



```
Файл  Правка  Вид  Закладки  Настройка  Справка
asmatveeva@dk3n54 ~ $ touch common.h server.c client.c Makefile
asmatveeva@dk3n54 ~ $ emacs &
```

Figure 3.1: Создание файлов и открытие редактора

2. Далее я изменила коды программ, представленных в тексте лабораторной работы. В файл common.h добавила стандартные заголовочные файлы unistd.h и time.h, необходимые для работы кодов других файлов. Common.h предназначен для заголовочных файлов, чтобы в остальных программах их не прописывать каждый раз. (рис. 3.2):

A screenshot of a code editor window with a menu bar (File, Edit, Options, Buffers, Tools, C, Help) and a text area containing C header file code. The code is for common.h and includes standard headers, defines FIFO_NAME and MAX_BUFF, and uses __COMMON_H__ for guard.

```
File Edit Options Buffers Tools C Help
/*
 * 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 <unistd.h>
#include <time.h>

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

#endif /* __COMMON_H__ */
```

Figure 3.2: Измененный файл common.h

В файл server.c добавила цикл while для контроля за временем работы сервера. Разница между текущим временем time(NULL) и временем начала работы clock_t start=time(NULL) (инициализация до цикла) не должна превышать 30 секунд (рис. 3.3, 3.4)

```

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)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    /* 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);
    }

    /* начало отсчёта времени */
    clock_t start = time(NULL);

    /* цикл работает, пока с момента отсчёта времени прошло не более 30 секунд */
    while(time(NULL)-start < 30)
    {

```

Figure 3.3: Измененный файл server.c


```

File Edit Options Buffers Tools C Help
while(time(NULL)-start < 30)
{
    /* 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);
    }
    exit(0);
}

```

Figure 3.4: Измененный файл server.c

В файл client.c добавила цикл, который отвечает за количество сообщений о текущем времени (4 сообщения), которое получается в результате выполнения команд на Рисунке 5 (*/текущее время/*) и команду sleep(5) для приостановки работы клиента на 5 секунд. (рис. 3.5) (рис. 3.6)

```

File Edit Options Buffers Tools C Help
/*
 * 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"

int main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen;

    /* баннер */
    printf("FIFO Client...\n");

    /* цикл, отвечающий за отправку сообщения о текущем времени */
    for (int i=0; i<4; i++)
    {
        /* получим доступ к FIFO */
        if ((writefd=open(FIFO_NAME, O_WRONLY))<0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                    __FILE__, strerror(errno));
            exit(-1);
            break;
        }
        /* текущее время */
        long int ttime=time(NULL);
        char* text=ctime(&ttime);

        /* передадим сообщение серверу */
        msglen=strlen(text);
        if (write(writefd, text, msglen)!=msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                    __FILE__, strerror(errno));
            exit(-2);
        }
    }
}

```

Figure 3.5: Измененный файл client.c

```

    /* приостановка работы на 5 секунд*/
    sleep(5);
}

    /* закроем доступ к FIFO */
    close(writefd);
    exit(0);
}

```

Figure 3.6: Измененный файл client.c

Makefile (файл для сборки) не изменяла. (рис. 3.7)

```

File Edit Options Buffers Tools Makefile Help

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

```

Figure 3.7: Makefile

3. После написания кодов, я, используя команду «make all», скомпилировала необходимые файлы (рис. 3.8)

```

asmatveeva@dk3n54 ~ $ make all
gcc server.c -o server
gcc client.c -o client

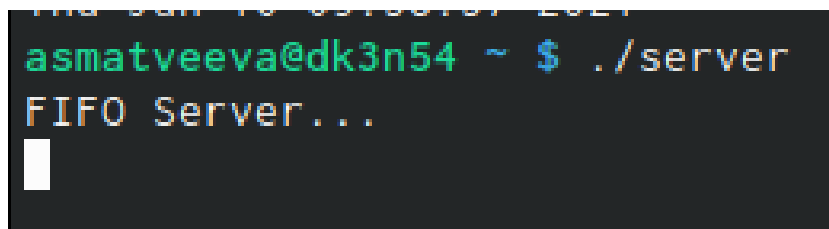
```

Figure 3.8: Компиляция

Далее я проверила работу написанного кода. Открыла 3 консоли терминала и запустила: в первом терминале – «./server», в остальных двух – «./client». В результате каждый терминал-клиент вывел по 4 сообщения. Спустя 30 секунд работа сервера была прекращена. Программа работает корректно. (рис. 3.9)

Figure 3.9: Проверка работы программы

Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запускать этот сервер снова, появится ошибка «Невозможно создать FIFO», так как у нас уже есть один канал.(рис. 3.10)

A terminal window with a dark background. The prompt is 'asmatveeva@dk3n54 ~ \$'. The user has entered './server' and the output is 'FIFO Server...'. A white cursor is visible on the line following the output.

```
asmatveeva@dk3n54 ~ $ ./server
FIFO Server...
```

Figure 3.10: Проверка работы программы

4 Выводы

В ходе выполнения данной лабораторной работы я приобрела практические навыки работы с именованными каналами.

5 Контрольные вопросы

- 1) 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.
- 2) Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс_1 | процесс_2 | процесс_3...
- 3) Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod », либо команду «mkfifo ».
- 4) Неименованный канал является средством взаимодействия между связанными процессами – родительским и дочерним. Родительский процесс создает канал при помощи системного вызова: «int pipe(int fd[2]);».

Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то этот массив содержит два файловых дескриптора. fd[0] является дескриптором для чтения из канала, fd[1] – дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует канал только для чтения, а другой – только для записи. Поэтому, если, например, через канал должны передаваться данные

из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой – в другую.

5) Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod`:

- «`int mkfifo(const char *pathname, mode_t mode);`», где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу),
- «`mknod (namefile, IFIFO | 0666, 0)`», где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу),
- «`int mknod(const char *pathname, mode_t mode, dev_t dev);`».

Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл канала уже существует, `mkfifo()` возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения.

6) При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. 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) 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=ERRPIPE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).
- 8) Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум `PIPE BUF` байтов данных. Предположим, процесс (назовем его А) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например, В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся X-Y байтов данных в канал. В результате данные в канал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.
- 9) Функция `write` записывает байты `count` из буфера `buffer` в файл, связанный с `handle`. Операции `write` начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для до-

бавления, операции выполняются в конец файла. После осуществления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция `write` возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа `count` (например, когда размер для записи `count` байтов выходит за пределы пространства на диске). Возвращаемое значение `-1` указывает на ошибку; `errno` устанавливается в одно из следующих значений:

`EACCES` – файл открыт для чтения или закрыт для записи,

`EBADF` – неверный `handle`-р файла,

`ENOSPC` – на устройстве нет свободного места.

Единица в вызове функции `write` в программе `server.c` означает идентификатор (дескриптор потока) стандартного потока вывода.

10) Прототип функции `strerror`: «`char * strerror(int errornum);`».

Функция `strerror` интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента – `errornum`, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет – использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции `strerror`.

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.

6 Библиография

1. https://esystem.rudn.ru/pluginfile.php/1142102/mod_resource/content/1/013-ipc-fifo.pdf
2. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
3. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)
4. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)