МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: Межпроцессные взаимодействия. Разделение памяти, семафоры, сокеты, каналы.

Студентка гр. 2384 _____ Соц Е.А.

Преподаватель _____ Душутина Е.В.

Санкт-Петербург
2024

Цель работы.

Целью данной работы является изучение средств межпроцессного взаимодействия в ОС семейства Unix

Задание.

8. Каналы: реализуйте pipe и

9. fifo. Приведите в отчете фрагмент файла, содержащего ограничения для IPC (для каналов и сообщений).

10. Осуществите передачу информации посредством обмена сообщениями по принципу «почтового ящика», т.е. не синхронизируя отправителя и получателя (без ожидания доставки).

11. Организуйте обмен сообщениями так, чтобы некоторому событию соответствовал отдельный тип сообщения. (Для реализации можно, например, использовать функции eventfd, poll)

12. Выполните передачу информации локально посредством сокетов по TCP/IP,

13. а затем в сетевом режиме (посредством сокетов по TCP/IP)

14. Организуйте взаимодействие с 10,100 и 1000 клиентами в клиент-серверном приложении (посредством сокетов). Оцените ограничения

15. Выполните аналогичное взаимодействие на основе UDP,

16. экспериментально продемонстрируйте разницу между TCP и UDP реализациями

17. Обеспечьте разделение памяти между независимыми процессами и необходимую синхронизацию для эффективного взаимодействия

*Задания 11, 14, 16 – для претендующих на «отлично»

Выполнение работы.

Информация о системе:

Linux katya 6.5.0-28-generic #29~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Thu Apr 4 14:39:20 UTC 2 x86_64 x86_64 x86_64 GNU/Linux

Задание 8. Каналы: реализуйте pipe

Неименованные каналы (или unnamed pipes) являются одним из способов межпроцессного взаимодействия (IPC) в операционных системах Unix и Linux. Они позволяют процессам обмениваться данными без необходимости использования файловой системы. Неименованный канал представляет собой пару конечных точек (файловых дескрипторов), одна из которых используется для записи данных, а другая — для их чтения. Эти конечные точки могут быть доступны только внутри процесса или между родительским и дочерним процессами.

```
(base) katya@katya:~/os/lb6$ gcc task8.c -o task8
(base) katya@katya:~/os/lb6$ cat task8.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define DEF_F_R "8from.txt"
#define DEF_F_W "8to.txt"


int main(int argc, char** argv) {
    char fileToRead[32];
    char fileToWrite[32];
    if(argc < 3 ){
                                printf("Using    default    fileNames
'%s','%s'\n",DEF_F_R,DEF_F_W);
        strcpy(fileToRead,DEF_F_R);
        strcpy(fileToWrite,DEF_F_W);
    }
    else{
        strcpy(fileToRead, argv[1]);
        strcpy(fileToWrite, argv[2]);
    }
    int filedes[2];
    if(pipe(filedes) < 0){
```

```
        printf("Father: can't create pipe\n");
        exit(1);
    }
    printf("pipe is successfully created\n");

        //процесс-сын
    if(fork() == 0){
        //закрытие дескриптора канала на чтение
        close(filedes[0]);

        // открытие файла на чтение
        FILE* f = fopen(fileToRead, "r");
        if(!f){
            printf("Son: can't open file %s\n", fileToRead);
            exit(1);
        }
        char buf[100];
        int res;
        while(!feof(f)){
            // чтение данных из файла
            res = fread(buf, sizeof(char), 100, f);
            // запись прочитанной строки в канал
            write(filedes[1], buf, res);
        }
        // закрытие файла из канала
        fclose(f);
        close(filedes[1]);
        return 0;
    }

        // процесс-родитель
    //закрытие дескриптора канала на запись
    close(filedes[1]);
    // открытие файла для записи
    FILE *f = fopen(fileToWrite, "w");
    if(!f){
        printf("Father: can't open file %s\n", fileToWrite);
        exit(1);
    }
    char buf[100];
    int res;
    while(1){
        // чтение из канала строки
        bzero(buf, 100);
        res = read(filedes[0], buf, 100);
        if(!res) break;
        printf("Read from pipe: %s\n", buf);
        // запись прочитанной строки в файл
        fwrite(buf, sizeof(char), res, f);
    }
    fclose(f);
    close(filedes[0]);
    return 0;
}
(base) katya@katya:~/os/lb6$ gcc task8.c -o task8
(base) katya@katya:~/os/lb6$ ./task8
```

```
Using default fileNames '8from.txt','8to.txt'
pipe is successfully created
Read from pipe: 1 Sots
2 Ekateryna
3 2384
4 lab rab 6

(base) katya@katya:~/os/lb6$ cat 8from.txt
1 Sots
2 Ekateryna
3 2384
4 lab rab 6
(base) katya@katya:~/os/lb6$ cat 8to.txt
1 Sots
2 Ekateryna
3 2384
4 lab rab 6
```

Этот пример демонстрирует использование неименованного канала (pipe) для передачи данных между родительским и дочерним процессами. Основная идея заключается в том, чтобы один процесс читал данные из файла и отправлял их в канал, а другой процесс читал эти данные из канала и записывал их в другой файл.

С помощью функции pipe(filedes) создаётся неименованный канал, который возвращает пару дескрипторов: filedes[0] для чтения и filedes[1] для записи. Если канал не может быть создан, программа выводит сообщение об ошибке и завершается.

Задание 9. fifo. Приведите в отчете фрагмент файла, содержащего ограничения для IPC (для каналов и сообщений).

Именованные каналы, также известные как FIFO (First In First Out), представляют собой метод межпроцессного взаимодействия (IPC), который является расширением традиционного понятия канала в Unix. В отличие от неименованных каналов, которые существуют только во время жизни процесса, именованные каналы могут существовать на протяжении

всего времени работы системы и даже после завершения всех процессов, которые с ними работали.

```
(base) katya@katya:~/os/lb6$ cat 9server.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define DEF_FILENAME "8from.txt"

int main(int argc, char **argv){
    char fileName[30];
    if(argc < 2){
        printf("Using default file name '%s'\n", DEF_FILENAME);
        strcpy(fileName, DEF_FILENAME);
    }else{
        strcpy(fileName, argv[1]);
    }

    // создание именованных каналов
    mknod("channel_write", S_IFIFO | 0666, 0);
    mknod("channel_read", S_IFIFO | 0666, 0);

    //открытие первого канала
    int chan1 = open("channel_write", O_WRONLY);
    if(chan1 == -1){
        printf("Can't open channel for writing\n");
        exit(0);
    }
    // открытие второго канала
    int chan2 = open("channel_read", O_RDONLY);
    if(chan2 == -1){
        printf("Can't open channel for reading\n");
        exit(0);
    }

    // запись имени файла в первый канал
    write(chan1, fileName, strlen(fileName));
    // чтение содержимого файла из второго канала
    char buf[100];
    for (;;){
        bzero(buf, 100);
        if(read(chan2, buf, 100) <= 0) break;
        printf("Part of file: %s\n", buf);
    }

    // закрытие каналов
    close(chan1);
    close(chan2);
```

```
    unlink("channel_write");
    unlink("channel_read");
    return 0;
}(base) katya@katya:~/os/lb6$ cat 9client.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


int main(int argc, char** argv){
      // открытие канала для чтения, где записано имя канала для
записи
    int chan1 = open("channel_write", O_RDONLY);
    if(chan1 == -1){
        printf("Can't open channel for reading\n");
        exit(0);
    }
    // открытие канала для записи
    int chan2 = open("channel_read", O_WRONLY);
    if(chan2 == -1){
        printf("Can't open channel for reading\n");
        exit(0);
    }

    // чтение имени файла из первого канала
    char fileName[100];
    bzero(fileName, 100);
    int res = read(chan1, fileName, 100);
    if(res <= 0){
        printf("Can't read filename from channel1\n");
        exit(0);
    }

    // открытие файла на чтение
    FILE *f = fopen(fileName, "r");
    if(!f){
        printf("Can't open file %s\n", fileName);
        exit(0);
    }
    char buf[100];
    while(!feof(f)){
        // чтение данных из файла
        int res = fread(buf, sizeof(char), 100, f);
        // запись их в канал
        write(chan2, buf, res);
    }
    fclose(f);
    // закрытие каналов
    close(chan1);
    close(chan2);
    return 0;
}(base) katya@katya:~/os/lb6$ gcc 9server.c -o 9server
```

```
(base) katya@katya:~/os/lb6$ gcc 9client.c -o 9client
(base) katya@katya:~/os/lb6$ ./9server
Using default file name '8from.txt'
Part of file: 1 Sots
2 Ekateryna
3 2384
4 lab rab 6

(base) katya@katya:~/os/lb6$ ./9client
```

В программе сервера происходит следующее:

1) Создание именованных каналов с помощью функции mknod: для записи channel_write и для чтения channel_read.

2) Открытие каналов для записи и чтения соответственно.

3) Запись имени файла в канал. Это предполагает, что client будет читать это имя из канала и использовать его для дальнейшей работы.

4) Чтение содержимого файла из канала.

5) Закрытие каналов и удаление.

В программе клиента:

1) Открытие каналов: предполагается, что другой процесс уже создал эти каналы и готов к обмену данными.

2) Чтение имени файла из канала для чтения. Это имя файла было ранее записано другим процессом в канал для записи.

3) Открытие файла для чтения, используя прочитанное имя файла.

4) Чтение данных из файла и запись в канал для записи. Это делается в цикле, пока не будут полностью прочитаны все данные из файла. После завершения чтения файла файл закрывается.

5) Закрытие каналов.

Ограничения для IPC для сообщений:

```
(base) katya@katya:~/os/lb6$ ipcs -l
```

```
------ Лимиты сообщений --------
максимум очередей для всей системы = 32000
максимальный размер сообщения (байты) = 8192
максимальный по умолчанию размер сообщения (байты) = 16384

------ Пределы совм. исп. памяти --------
макс. количество сегментов = 4096
макс. размер сегмента (килобайты) = 18014398509465599
max total shared memory (kbytes) = 18446744073709551612
мин. размер сегмента (байты) = 1

------ Пределы семафоров --------
максимальное количество массивов = 32000
максимум семафоров на массив = 32000
максимум семафоров на всю систему = 1024000000
максимум операций на вызов семафора = 500
максимальное значение семафора = 32767
```

Ограничения для IPC для каналов:

```
(base) katya@katya:~/os/lb6$ cat 9lim.c
#include <unistd.h>
#include <stdio.h>
#include <limits.h>

int main() {
    printf("PIPE_BUF: %d\n", PIPE_BUF);
    long open_max = sysconf(_SC_OPEN_MAX);
    printf("OPEN_MAX: %ld\n", open_max);
    return 0;
}
(base) katya@katya:~/os/lb6$ ./9lim
PIPE_BUF: 4096
OPEN_MAX: 1024
(base) katya@ka
```

Задание 10. Осуществите передачу информации посредством обмена сообщениями по принципу «почтового ящика», т.е. не синхронизируя отправителя и получателя (без ожидания доставки).

Очередь сообщений находится в адресном пространстве ядра и имеет ограниченный размер. В отличие от каналов, которые обладают теми же самыми свойствами, очереди сообщений сохраняют границы сообщений. Это значит, что ядро ОС гарантирует, что сообщение, поставленное в очередь, не смешается с предыдущим или следующим сообщением при чтении из очереди. Кроме того, с каждым сообщением связывается его тип.

Для записи сообщения в очередь не требуется наличия ожидающего его процесса в отличие от неименованных каналов и FIFO, в которые нельзя произвести запись, пока не появится считывающий данные процесс. Поэтому процесс может записать в очередь какие-то сообщения, после чего они могут быть получены другим процессом в любое время, даже если первый завершит свою работу. С завершением процесса-источника данные не исчезают (данные, остающиеся в именованном или неименованном канале, сбрасываются, после того как все процессы закроют его)

```
(base) katya@katya:~/os/lb6$ cat task10.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <signal.h>
#include <string.h>

typedef struct
{
    long type;
    char buf[64];
} Message;


int main(int argc, char **argv)
{
    // id очереди по ее ключу
    int qid = msgget((key_t)atoi(argv[1]), IPC_CREAT | 0666);

    // отправка сообщений в ящик
    Message msg1 = {1, "first message"};
    Message msg2 = {2, "second message"};
    msgsnd(qid, &msg1, strlen(msg1.buf) + 1, 0);
    msgsnd(qid, &msg2, strlen(msg2.buf) + 1, 0);

    // получение сообщений из очереди
    // Message rmsg1;
    // Message rmsg2;
    // msgrcv(qid, &rmsg1, 64, 1, 0);
    // msgrcv(qid, &rmsg2, 64, 2, 0);
     // printf("message with type %ld recieved: %s\n", rmsg1.type,
rmsg1.buf);
     // printf("message with type %ld recieved: %s\n", rmsg2.type,
rmsg2.buf);
```

```
    // удаление очереди
    // msgctl(qid, IPC_RMID, 0);
    return 0;
}
(base) katya@katya:~/os/lb6$ gcc task10.c -o task10
(base) katya@katya:~/os/lb6$ ./task10 1000
(base) katya@katya:~/os/lb6$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения
0x000003e8 1          katya      666        29           2

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x00000000 9              katya        600           524288       2
назначение
0x00000000 14             katya        600           524288       2
назначение
0x00000000 20             katya        600           524288       2
назначение
0x00000000 24             katya        600           524288       2
назначение
0x00000000 25             katya        600           4194304      2
назначение
0x00000000 28             katya        600           524288       2
назначение
0x00000000 30             katya        606           6304800      2
назначение
0x00000000 31             katya        606           6304800      2
назначение
0x00000000 40             katya        600           524288       2
назначение
0x00000000 43             katya        600           524288       2
назначение
0x00000000 45             katya        600           4194304      2
назначение

------ Массивы семафоров --------
ключ   semid      владелец права nsems



(base) katya@katya:~/os/lb6$ cat task10.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <signal.h>
#include <string.h>

typedef struct
{
    long type;
    char buf[64];
```

```
} Message;


int main(int argc, char **argv)
{
    // id очереди по ее ключу
    int qid = msgget((key_t)atoi(argv[1]), IPC_CREAT | 0666);

    // отправка сообщений в ящик
    // Message msg1 = {1, "first message"};
    // Message msg2 = {2, "second message"};
    // msgsnd(qid, &msg1, strlen(msg1.buf) + 1, 0);
    // msgsnd(qid, &msg2, strlen(msg2.buf) + 1, 0);

    // получение сообщений из очереди
    Message rmsg1;
    Message rmsg2;
    msgrcv(qid, &rmsg1, 64, 1, 0);
    msgrcv(qid, &rmsg2, 64, 2, 0);
      printf("message with type %ld recieved: %s\n", rmsg1.type,
rmsg1.buf);
      printf("message with type %ld recieved: %s\n", rmsg2.type,
rmsg2.buf);

    // удаление очереди
    // msgctl(qid, IPC_RMID, 0);
    return 0;
}
(base) katya@katya:~/os/lb6$ gcc task10.c -o task10
(base) katya@katya:~/os/lb6$ ./task10 1000
message with type 1 recieved: first message
message with type 2 recieved: second message
(base) katya@katya:~/os/lb6$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения
0x000003e8 1          katya      666        0            0

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x00000000 9              katya        600          524288       2
назначение
0x00000000 14             katya        600          524288       2
назначение
0x00000000 20             katya        600          524288       2
назначение
0x00000000 24             katya        600          524288       2
назначение
0x00000000 25             katya        600          4194304      2
назначение
0x00000000 28             katya        600          524288       2
назначение
0x00000000 30             katya        606          6304800      2
назначение
0x00000000 31             katya        606          6304800      2
назначение
```

```
0x00000000 40             katya        600          524288       2
назначение
0x00000000 43             katya        600          524288       2
назначение
0x00000000 45             katya        600          4194304      2
назначение

------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

```
(base) katya@katya:~/os/lb6$ cat task10.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <signal.h>
#include <string.h>

typedef struct
{
    long type;
    char buf[64];
} Message;


int main(int argc, char **argv)
{
    // id очереди по ее ключу
    int qid = msgget((key_t)atoi(argv[1]), IPC_CREAT | 0666);

    // отправка сообщений в ящик
    // Message msg1 = {1, "first message"};
    // Message msg2 = {2, "second message"};
    // msgsnd(qid, &msg1, strlen(msg1.buf) + 1, 0);
    // msgsnd(qid, &msg2, strlen(msg2.buf) + 1, 0);

    // получение сообщений из очереди
    // Message rmsg1;
    // Message rmsg2;
    // msgrcv(qid, &rmsg1, 64, 1, 0);
    // msgrcv(qid, &rmsg2, 64, 2, 0);
     // printf("message with type %ld recieved: %s\n", rmsg1.type,
rmsg1.buf);
     // printf("message with type %ld recieved: %s\n", rmsg2.type,
rmsg2.buf);

    // удаление очереди
    msgctl(qid, IPC_RMID, 0);
    return 0;
}
(base) katya@katya:~/os/lb6$ gcc task10.c -o task10
(base) katya@katya:~/os/lb6$ ./task10 1000
```

```
(base) katya@katya:~/os/lb6$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x00000000 9              katya        600           524288       2
назначение
0x00000000 14             katya        600           524288       2
назначение
0x00000000 20             katya        600           524288       2
назначение
0x00000000 24             katya        600           524288       2
назначение
0x00000000 25             katya        600           4194304      2
назначение
0x00000000 28             katya        600           524288       2
назначение
0x00000000 30             katya        606           6304800      2
назначение
0x00000000 31             katya        606           6304800      2
назначение
0x00000000 40             katya        600           524288       2
назначение
0x00000000 43             katya        600           524288       2
назначение
0x00000000 45             katya        600           4194304      2
назначение

------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

Сначала программа создает очередь сообщений с помощью функции msgget(). Эта функция принимает ключ очереди (преобразованный из строки аргумента командной строки) и флаги, указывающие на то, что очередь должна быть создана (IPC_CREAT) и иметь определенные права доступа (0666). Результатом является идентификатор очереди.

Затем программа создает два объекта типа Message. Эти сообщения отправляются в очередь с помощью функции msnd(). Первый аргумент этой функции — идентификатор очереди, второй — адрес сообщения, третье — количество байтов сообщения, а четвертый — флаги, указывающие на поведение при переполнении очереди (0 означает стандартное поведение).

Программа получает два сообщения из очереди с помощью функции msgrcv(), которая принимает идентификатор очереди, адрес места назначения для сообщения, максимальный размер сообщения, тип сообщения для получения и флаги.

После завершения работы с очередью она удаляется с помощью функции msgctl().

Промежуточные выводы ipcs отображают состояние созданной очереди.

Задание 12. Выполните передачу информации локально посредством сокетов по TCP/IP,

Сокеты представляют собой мощный инструмент для межпроцессного взаимодействия (IPC) в Unix-подобных операционных системах. Они позволяют процессам обмениваться данными напрямую, минуя файловую систему, что обеспечивает высокую скорость и эффективность коммуникации.

Так как сокеты удобное средство для реализации клиент-серверных приложений, они располагаются на хосте на определенном порте, о котором знает запрашивающее приложение/процесс, после этого в случае TCP сокетов для каждого установленного соединения выделяется отдельный сокет для общения с клиентом, работу с которым удобно производить в отдельном потоке.

Сначала серверный сокет привязывается к адресу и слушает на нем входящие подключения, клиент в свою очередь шлет на localhost на указанный порт сообщения и получает ответы.

При получении сообщения используется флаг MSG_WAITALL, чтобы получить всю запрошенную длину, которая еще, быть может, не дошла, но TCP гарантирует ее доставку.

```
      (base) katya@katya:~/os/lb6$ cat 12client.c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define SERVER_PORT 8888
#define SERVER_IP "127.0.0.1"

int readFix(int sock, char *buf, int size, int flags)
{
    // читаем "заголовок" - сколько байт составляет наше сообщение
    unsigned msg_len = 0;
       int res = recv(sock, &msg_len, sizeof(unsigned), flags |
MSG_WAITALL);
    if (res <= 0)
        return res;
    // читаем само сообщение
    return recv(sock, buf, msg_len, flags | MSG_WAITALL);
}

int sendFix(int sock, char *buf, int flags)
{
    // шлем число байт в сообщение
    unsigned msg_len = strlen(buf);
    int res = send(sock, &msg_len, sizeof(unsigned), flags);
    if (res <= 0)
        return res;
    return send(sock, buf, msg_len, flags);
}


int main()
{
    // создаем  сокет, подключаемся к серверу
    struct sockaddr_in peer;
    peer.sin_family = AF_INET;
    peer.sin_port = htons(SERVER_PORT);
    peer.sin_addr.s_addr = inet_addr(SERVER_IP);
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    connect(sock, (struct sockaddr*)&peer, sizeof(peer));

    char buf[100];
    while (1) {
```

```
        printf("Сообщение для отправки на сервер:\n");
        bzero(buf, 100);
        fgets(buf, 100, stdin);
        buf[strlen(buf)-1] = '\0';
        if(strlen(buf) == 0){
            printf("Завершение работы клиетна\n");
            return 0;
        }
        int res = sendFix(sock, buf, 0);
        if(res<0){
            perror("Ошибка отправки\n");
            exit(1);
        }
        bzero(buf, 100);
        res = readFix(sock, buf, 100, 0);
        printf("Ответ сервера: %s\n", buf);
    }

}
```

Функция sendFix перед посылкой собственно данных посылает «заголовок» - количество байт в посылке. Функция recvFix вначале принимает этот «заголовок», и вторым вызовом recv считывает переданное количество байт. Считать ровно то, количество байт, которое указано в аргументе функции recv, позволяет флаг MSG_WAITALL. Если его не использовать и данных в буфере недостаточно, то будет прочитано меньшее количество.

```
(base) katya@katya:~/os/lb6$ ./12client
Сообщение для отправки на сервер:
hello
Ответ сервера: hello
Сообщение для отправки на сервер:
katya sots
Ответ сервера: katya sots
Сообщение для отправки на сервер:
2384!
Ответ сервера: 2384!
Сообщение для отправки на сервер:

Завершение работы клиетна

(base) katya@katya:~/os/lb6$ ./task12
Пустое сообщение от клиента, поток завершается
```

Задание 13. а затем в сетевом режиме (посредством сокетов по TCP/IP)

Утилиты ifconfig и nmcli демонстрируют информацию о сетевых интерфейсах определяется ip-адрес роутера, на который высылаются пакеты из клиентского приложения, а тот маршрутизирует их обратно, на компьютер, где они обрабатываются сервером.

```
(base) katya@katya:~/os/lb6$ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Локальная петля (Loopback))
        RX packets 17605  bytes 1764637 (1.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17605  bytes 1764637 (1.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
            inet 172.20.10.2   netmask 255.255.255.240   broadcast
172.20.10.15
          inet6 fe80::86ca:5cc3:df07:c629   prefixlen 64   scopeid
0x20<link>
        ether 14:13:33:05:6e:35  txqueuelen 1000  (Ethernet)
        RX packets 516708  bytes 515959642 (515.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 245491  bytes 45062149 (45.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

(base) katya@katya:~/os/lb6$ nmcli device status
DEVICE          TYPE      STATE          CONNECTION
wlp1s0          wifi      подключено     iPhone
p2p-dev-wlp1s0  wifi-p2p  отключено      --
lo              loopback  не настроенно  --


(base) katya@katya:~/os/lb6$ cat task13.c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define SERVER_PORT 8888

int readFix(int sock, char *buf, int size, int flags)
{
    unsigned msg_len = 0;
      int  res  =  recv(sock,  &msg_len,  sizeof(unsigned),  flags  |
MSG_WAITALL);
```

```
    if (res <= 0)
        return res;
    return recv(sock, buf, msg_len, flags | MSG_WAITALL);
}

int sendFix(int sock, char *buf, int flags)
{
    unsigned msg_len = strlen(buf);
    int res = send(sock, &msg_len, sizeof(unsigned), flags);
    if (res <= 0)
        return res;
    return send(sock, buf, msg_len, flags);
}

void* handler(void *args)
{
    // обработчик для отдельного клиента
    int sock = *(int*)args;
    char buf[100];
    while (1) {
        bzero(buf, 100);
        int res = readFix(sock, buf, 100, 0);
        if (res <= 0) {
                        puts("Пустое сообщение от клиента, поток
завершается");
            pthread_exit(NULL);
        }
        res = sendFix(sock, buf, 0);
        if (res <= 0) {
            perror("Отправка не удалась");
            pthread_exit(NULL);
        }
    }
}

int main()
{
    // создаем  TCP-сокет, слушающий SERVER_PORT
    struct sockaddr_in listener_info;
    listener_info.sin_family = AF_INET;
    listener_info.sin_port = htons(SERVER_PORT);
    listener_info.sin_addr.s_addr = htonl(INADDR_ANY);
    int sock = socket(AF_INET, SOCK_STREAM, 0);
                bind(sock,    (struct    sockaddr*)&listener_info,
sizeof(listener_info));
    listen(sock, 5);

    // обработка подключений
    while (1) {
        int client = accept(sock, NULL, NULL);
        pthread_t tid;
        pthread_create(&tid, NULL, handler, (void*)&client);
    }
    return 0;
}
(base) katya@katya:~/os/lb6$ cat 13client.c
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define SERVER_PORT 8888
#define SERVER_IP "172.20.10.2"

int readFix(int sock, char *buf, int size, int flags)
{
    unsigned msg_len = 0;
       int  res  =  recv(sock,  &msg_len,  sizeof(unsigned),  flags  |
MSG_WAITALL);
    if (res <= 0)
        return res;
    return recv(sock, buf, msg_len, flags | MSG_WAITALL);
}

int sendFix(int sock, char *buf, int flags)
{
    unsigned msg_len = strlen(buf);
    int res = send(sock, &msg_len, sizeof(unsigned), flags);
    if (res <= 0)
        return res;
    return send(sock, buf, msg_len, flags);
}

int main()
{
    // создаем сокет и подключаемся к серверу
    struct sockaddr_in peer;
    peer.sin_family = AF_INET;
    peer.sin_port = htons(SERVER_PORT);
    peer.sin_addr.s_addr = inet_addr(SERVER_IP);
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    connect(sock, (struct sockaddr*)&peer, sizeof(peer));


    char buf[100];
    while (1) {
        printf("Сообщение для отправки на сервер:\n");
        bzero(buf,100);
        fgets(buf, 100, stdin);
        buf[strlen(buf)-1] = '\0';
        if(strlen(buf) == 0) {
            puts("Завершение работы клиента");
            return 0;
        }
        int res = sendFix(sock, buf,0);
        if (res <= 0) {
            perror("Проблемы при отправке");
```

```
            exit(1);
        }
        bzero(buf, 100);
        res = readFix(sock, buf, 100, 0);
        printf("Ответ сервера: %s\n",buf);
    }
}
```

```
(base) katya@katya:~/os/lb6$ gcc task13.c -o task13
(base) katya@katya:~/os/lb6$ gcc 13client.c -o 13client
(base) katya@katya:~/os/lb6$ ./task13 &
[2] 27164
(base) katya@katya:~/os/lb6$ ./13client
Сообщение для отправки на сервер:
hello
Ответ сервера: hello
Сообщение для отправки на сервер:
lalalal
Ответ сервера: lalalal
Сообщение для отправки на сервер:
kdokosko
Ответ сервера: kdokosko
Сообщение для отправки на сервер:

Завершение работы клиента
(base) katya@katya:~/os/lb6$ Пустое сообщение от клиента, поток
завершается
```

Задание 15. Выполните аналогичное взаимодействие на основе UDP,

Был имплементирован UDP-сервер и клиент, которые успешно взаимодействовали между собой. В отличие от TCP не требуется использовать accept и connect, а вместо recv и send используется recvfrom и sendto, при этом sendto неблокирующий, потому что UDP не гарантирует доставку получателю.

```
(base) katya@katya:~/os/lb6$ cat 15server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_BUFFER_SIZE 1024
#define SERVER_PORT 8888
```

```
int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[MAX_BUFFER_SIZE];

    // Создание сокета
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // Настройка адреса сервера
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    // Привязка сокета к адресу
                bind(sockfd,    (struct    sockaddr*)&server_addr,
sizeof(server_addr));


    // Прием и отправка сообщений
    while (1) {
                ssize_t  bytes_received  =  recvfrom(sockfd,  buffer,
MAX_BUFFER_SIZE - 1, 0,
                                                                (struct
sockaddr*)&client_addr, &addr_len);
        if (bytes_received <= 0) continue;

        buffer[bytes_received] = '\0';
        printf("Полученное сообщение от клиента: %s\n", buffer);

            sendto(sockfd, "Сообщение получено", strlen("Сообщение
получено"), 0,
                (struct sockaddr*)&client_addr, addr_len);
    }

    close(sockfd);
    return 0;
}

(base) katya@katya:~/os/lb6$ cat 15client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_BUFFER_SIZE 1024
#define SERVER_PORT 8888
#define SERVER_IP "127.0.0.1"

int main(int argc, char *argv[]) {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[MAX_BUFFER_SIZE];
```

```
    int num_messages = 10; // Количество сообщений для отправки

    // Создание сокета
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    // Настройка адреса сервера
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    server_addr.sin_addr.s_addr = inet_addr(SERVER_IP);

    // Отправка сообщений
    for (int i = 1; i <= num_messages; i++) {
        sprintf(buffer, "Сообщение %d", i);
              sendto(sockfd, buffer, strlen(buffer), 0, (struct
sockaddr*)&server_addr, sizeof(server_addr));
    }

    // Прием ответов
    socklen_t addr_len = sizeof(server_addr);
    for (int i = 1; i <= num_messages; i++) {
              ssize_t bytes_received = recvfrom(sockfd, buffer,
MAX_BUFFER_SIZE - 1, 0,
                                                    (struct
sockaddr*)&server_addr, &addr_len);
        if (bytes_received > 0) {
            buffer[bytes_received] = '\0';
            printf("Ответ сервера: %s\n", buffer);
        } else {
            printf("Нет ответа %d\n", i);
        }
    }

    close(sockfd);

    return 0;
}

(base) katya@katya:~/os/lb6$ ss -lu
State    Recv-Q   Send-Q     Local Address:Port      Peer Address:Port
Process
UNCONN   0        0              0.0.0.0:53849          0.0.0.0:*
UNCONN   0        0           224.0.0.251:mdns          0.0.0.0:*
UNCONN   0        0           224.0.0.251:mdns          0.0.0.0:*
UNCONN   0        0              0.0.0.0:mdns          0.0.0.0:*
UNCONN   0        0         127.0.0.53%lo:domain        0.0.0.0:*
UNCONN   0        0              0.0.0.0:631           0.0.0.0:*
UNCONN   0        0                 [::]:mdns             [::]:*
UNCONN   0        0                 [::]:42328            [::]:*
(base) katya@katya:~/os/lb6$ ./15server &
[1] 19315
(base) katya@katya:~/os/lb6$ ss -lu
```

```
State    Recv-Q   Send-Q    Local Address:Port      Peer Address:Port
Process
UNCONN   0        0              0.0.0.0:53849          0.0.0.0:*
UNCONN   0        0            224.0.0.251:mdns         0.0.0.0:*
UNCONN   0        0            224.0.0.251:mdns         0.0.0.0:*
UNCONN   0        0              0.0.0.0:mdns           0.0.0.0:*
UNCONN   0        0         127.0.0.53%lo:domain        0.0.0.0:*
UNCONN   0        0              0.0.0.0:631            0.0.0.0:*
UNCONN   0        0              0.0.0.0:8888           0.0.0.0:*
UNCONN   0        0                 [::]:mdns              [::]:*
UNCONN   0        0                 [::]:42328             [::]:*
(base)  katya@katya:~/os/lb6$  Полученное  сообщение  от  клиента:
Сообщение 1
Полученное сообщение от клиента: Сообщение 2
Полученное сообщение от клиента: Сообщение 3
Полученное сообщение от клиента: Сообщение 4
Полученное сообщение от клиента: Сообщение 5
Полученное сообщение от клиента: Сообщение 6
Полученное сообщение от клиента: Сообщение 7
Полученное сообщение от клиента: Сообщение 8
Полученное сообщение от клиента: Сообщение 9
Полученное сообщение от клиента: Сообщение 10

base) katya@katya:~/os/lb6$ ./15client
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
Ответ сервера: Сообщение получено
```

Команда ss -lu в Linux используется для отображения списка активных UDP-соединений на системе.

Задание 17. Обеспечьте разделение памяти между независимыми процессами и необходимую синхронизацию для эффективного взаимодействия

Для решения поставленной задачи использовался механизм разделяемой памяти (shared memory) и семафоры.

Семафоры и разделяемая память зачастую работают вместе. Семафоры позволяют синхронизировать доступ к разделяемому ресурсу и

гарантировать «взаимное исключение» нескольких процессов при разделении ресурса (пока предыдущий процесс не закончит работу с ресурсом, следующий не начнет ее).

В программе-примере создавался сегмент разделяемой памяти в виде массива, куда могли писать процессы-писатели, последний индекс, с которым происходило взаимодействие записывался в конце массива.

При помощи заранее прописанных структур и функции semop происходило взятие/возврат семафоров, которых было 3 штуки.

Видно, что все записанные числа были считаны процессом-читателям, из тех ячеек, куда и записывались, то есть не произошло перезаписи или выхода за пределы памяти.

Удаление сегмента и удаление семафора реализовано через обработчик сигнала.

```
(base) katya@katya:~/os/lb6$ cat 17server.c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "shm.h"

int *buf;
int shmid;
int semaphore;

void handler(int sig)
{
    // удаление сегмента разделяемой памяти и семафора
    shmdt(buf);
    shmctl(shmid, IPC_RMID, 0);
    semctl(semaphore, 0, IPC_RMID);
}

int main()
{
    signal(SIGINT, handler);
```

```
    // создаем участок разделяемой памяти
     shmid = shmget(KEY, (BUF_SIZE + 1) * sizeof(int), IPC_CREAT |
0666);
    // получаем адрес выделенной разделяемой памяти
    buf = (int*)shmat(shmid, 0, 0);

    // создаем массив из 3 семафоров
    // 0 - число свободных ячеек
    // 1 - число занятых ячеек
    // 2 - работа с памятью
    semaphore = semget(KEY, 3, IPC_CREAT | 0666);
    // инициализируем память -1 и говорим, что она свободна
    for (int i = 0; i < BUF_SIZE + 1; ++i)
        buf[i] = -1;
    // устанавливаем все ячейки свободными и разблокируем память
    semop(semaphore, set_free, 1);
    semop(semaphore, mem_unlock, 1);
    puts("Нажать кнопку для начала работы");
    getchar();
    for (int i =0; i < 20; ++i) {
        // ждем пока будет хотя бы одна непустая ячейка
        semop(semaphore, wait_not_empty, 1);
        // ждем возможности взаимодействовать с памятью
        semop(semaphore, mem_lock, 1);
        // читаем информацию из памяти
        // требуемые индекс лежит после основного массива
        int res = buf[buf[BUF_SIZE]];
        buf[BUF_SIZE] = buf[BUF_SIZE] - 1;
        printf("Получен %d из ячейки %d\n", res, buf[BUF_SIZE]+1);
        // освобождаем память и увеличиваем число пустых ячеек
        semop(semaphore, mem_unlock, 1);
        semop(semaphore, release_empty, 1);
    }
    kill(getpid(), SIGINT);
    return 0;
}
(base) katya@katya:~/os/lb6$ cat 17client.c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "shm.h"

int *buf;

int main()
{
    // присоединились к разделяемой памяти и семафорам
    int shmid = shmget(KEY, (BUF_SIZE+1)*sizeof(int), 0666);
```

```
    buf = (int*)shmat(shmid, 0, 0);
    int semaphore = semget(KEY, 3, 0666);
    puts("Нажать кнопку для начала работы");
    getchar();
    int val = 0;
    for (int i = 0; i < 10; ++i) {
        // ждем свободных ячеек
        semop(semaphore, wait_not_full, 1);
        // ждем доступа к разделяемой памяти
        semop(semaphore, mem_lock, 1);
        // пишем в ячейку
        ++buf[BUF_SIZE];
        printf("Пишем %d в ячейку %d\n", val, buf[BUF_SIZE]);
        buf[buf[BUF_SIZE]] = val++;
        // освобождаем доступ к памяти
        semop(semaphore, mem_unlock, 1);
        // увеличиваем счетчик занятых ячеек
        semop(semaphore, release_full, 1);
    }
    shmdt(buf);
    return 0;
}
(base) katya@katya:~/os/lb6$ gcc 17server.c -o 17server
(base) katya@katya:~/os/lb6$ gcc 17client.c -o 17client
(base) katya@katya:~/os/lb6$ cat shm.h
#define KEY 2004
#define BUF_SIZE 15

static struct sembuf set_free[1] = { 0, BUF_SIZE, 0 };

static struct sembuf wait_not_full[1] = { 0, -1, 0 };
static struct sembuf wait_not_empty[1] = { 1, -1, 0 };

static struct sembuf release_empty[1] = { 0, 1, 0 };
static struct sembuf release_full[1] = { 1, 1, 0 };

static struct sembuf mem_lock[1] = { 2, -1, 0 };
static struct sembuf mem_unlock[1] = { 2, 1, 0 };


(base) katya@katya:~/os/lb6$ ./17server
Нажать кнопку для начала работы

Получен 5 из ячейки 5
Получен 6 из ячейки 5
Получен 7 из ячейки 5
Получен 8 из ячейки 5
Получен 9 из ячейки 5
Получен 4 из ячейки 4
Получен 3 из ячейки 3
Получен 2 из ячейки 2
Получен 1 из ячейки 1
Получен 0 из ячейки 0
(base) katya@katya:~/os/lb6$ ./17client
Нажать кнопку для начала работы
```

```
Пишем 0 в ячейку 0
Пишем 1 в ячейку 1
Пишем 2 в ячейку 2
Пишем 3 в ячейку 3
Пишем 4 в ячейку 4
Пишем 5 в ячейку 5
Пишем 6 в ячейку 5
Пишем 7 в ячейку 5
Пишем 8 в ячейку 5
Пишем 9 в ячейку 5
```

Литература

1. https://elib.spbstu.ru/dl/2/s17-72.pdf/en/view - Методическое пособие “Межпроцессные взаимодействия в операционных системах”, Душутина Е.В.
2. https://www.opennet.ru/base/dev/ipc_msg.txt.html - Статья про очередь сообщений
3. https://www.opennet.ru/docs/RUS/xtoolkit/x-1.html#x-1-7-3-2 - семафоры