

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный ядерный исследовательский университет
«МИФИ» (НИЯУ МИФИ)
Институт Интеллектуальных Кибернетических
Систем Кафедра Кибернетики

Лабораторная работа №3

По курсу «Разработка программного обеспечения ОС UNIX»

Работу выполнил студент группы Б17-511:

Павлов И.Е.

Проверил:

Ктитров С.В.

Москва 2020

Постановка задачи

Тема «Сокеты, разделяемая память»

Разработать программу, синхронизирующую содержимое разных экземпляров разделяемой памяти. При внесении изменения в один из экземпляров, другие должны обновиться. Разработать программу чтения содержимого экземпляра разделяемой памяти (с любого компьютера) с учетом возможного ее изменения в процессе чтения. Программа должна собираться из нескольких файлов с использованием make.

Объяснение логики

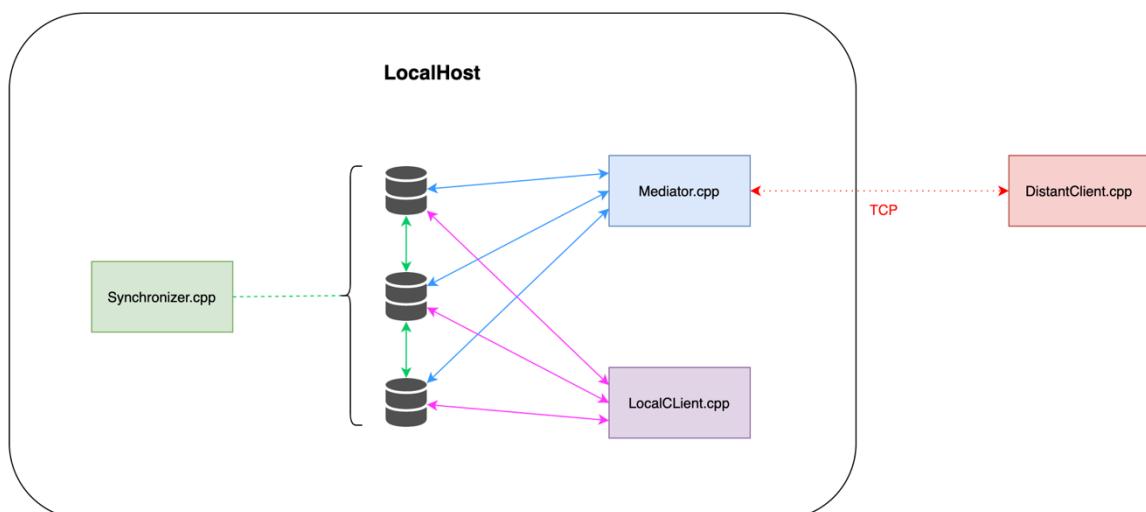
Три экземпляра разделяемой памяти создаются программой *Synchronizer*. Также *Synchronizer* реагирует на изменение любого экземпляра разделяемой памяти: если значение любого экземпляра было изменено, *Synchronizer* меняет значение остальных на такое же.

Программа *LocalClient* может подключаться к любому экземпляру разделяемой памяти. Она может работать с разделяемой памятью:

1. Читать содержимое любого экземпляра разделяемой памяти (если ввести кодовое слово *read*);
2. Изменять содержимое любого экземпляра разделяемой памяти (если ввести новое значение (нельзя: *read, exit*));
3. Завершить работу *Synchronizer*'а (если ввести кодовое слово *exit*).

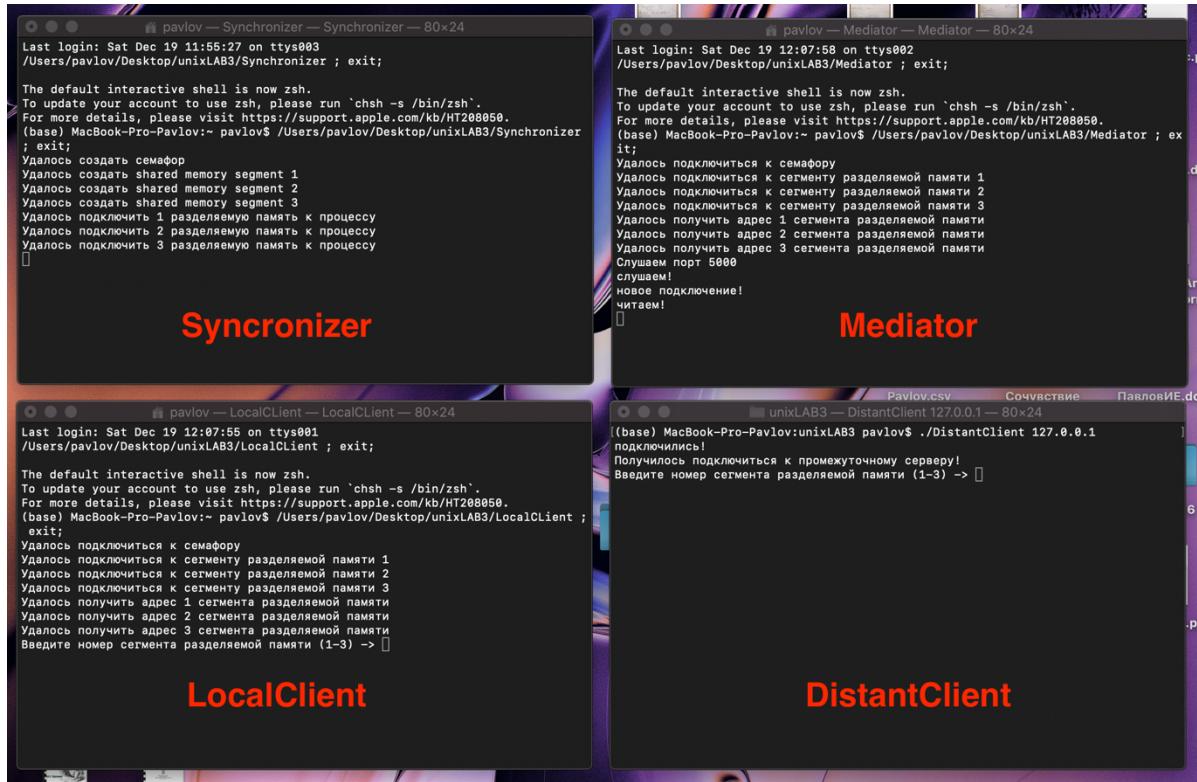
Программа *DistantClient* имеет тот же функционал, что и *LocalClient*, только может находиться на любом другом компьютере сети. *DistantClient* подключается по TCP к промежуточному серверу *Mediator* и отправляет ему новое значение, либо кодовое слово (*read, exit*). Если кодовое слово – *read*, то *DistantClient* ждет на прием данные. (*Запуск из командной строки – аргумент ip компьютера, на котором расположены экземпляры разделяемой памяти*).

Программа *Mediator* – промежуточный TCP сервер между экземплярами разделяемой памяти на *LocalHost* и *DistantClient* на любом другом компьютере сети. *Mediator* работает с разделяемой памятью аналогично *LocalClient*, за исключением того, что команды считывает не из командной строки, а от *DistantClient* по TCP.

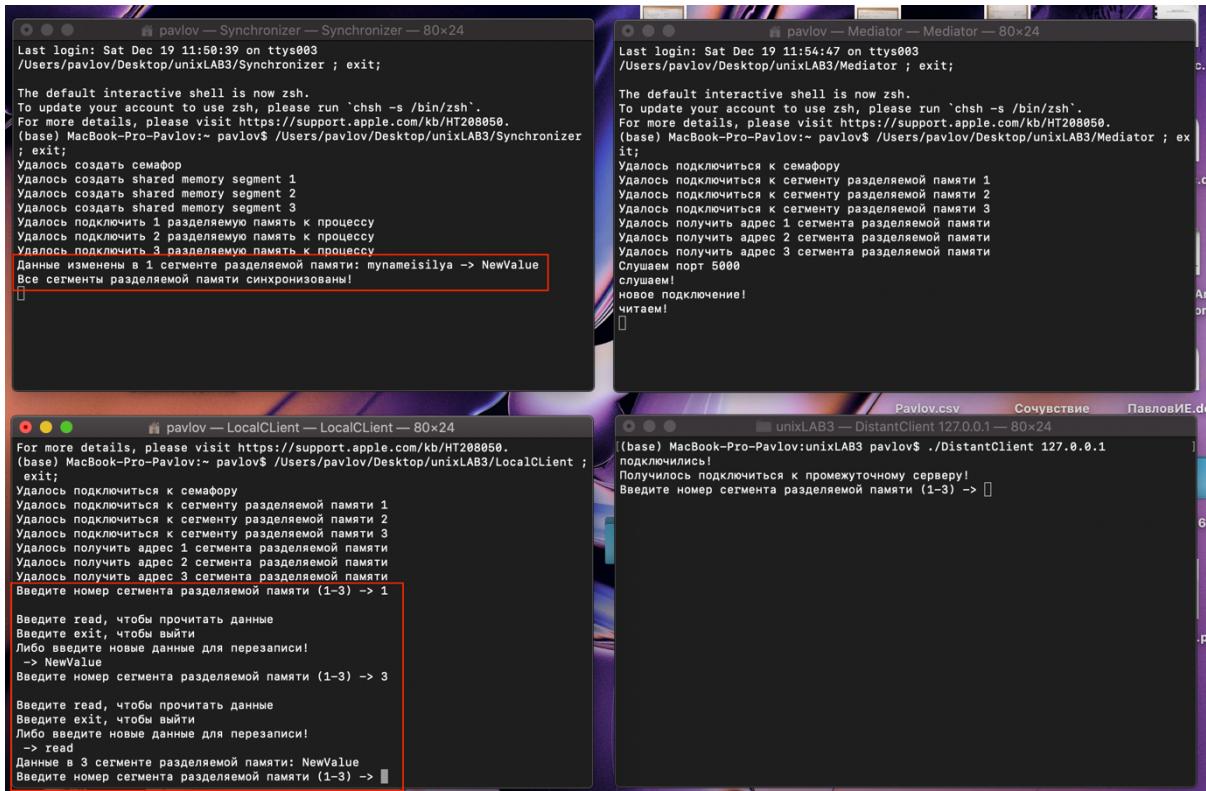


Пример работы

Запускаем все программы



Изначально данные были равны “mynameisilya”. Меняем через *LocalClient* на “*NewValue*” и потом читаем их.



Теперь меняем данные через *DistantClient* (“*NewValueNewValueFromDistant*”)

The image shows four terminal windows arranged in a 2x2 grid, illustrating the interaction between a Synchronizer, Mediator, LocalClient, and a distant client.

- Synchronizer Terminal:** Shows the creation of three shared memory segments (1, 2, 3) and their synchronization.
- Mediator Terminal:** Shows the Synchronizer sending a message to the Mediator, which then receives the update from the distant client.
- LocalClient Terminal:** Shows the LocalClient reading the shared memory and responding to the Mediator's request for new data.
- DistantClient Terminal:** Shows the DistantClient sending the updated value (00202) to the LocalClient.

Red boxes highlight the exchange of data between the LocalClient and the DistantClient terminals.

Теперь читаем эти данные через *DistantClient*

The image shows four terminal windows arranged in a 2x2 grid, illustrating the process of reading shared memory via DistantClient.

- Synchronizer Terminal:** Shows the creation of three shared memory segments (1, 2, 3) and their synchronization.
- Mediator Terminal:** Shows the Synchronizer sending a message to the Mediator, which then receives the read request from the distant client.
- LocalClient Terminal:** Shows the LocalClient responding to the read request by sending the current value (0019).
- DistantClient Terminal:** Shows the DistantClient receiving the value and printing it to the console.

Red boxes highlight the exchange of data between the LocalClient and the DistantClient terminals.

Листинг программ

makefile

```
CC=gcc
FLAGS=-stdc++
SYNCHRONIZER_SRC=Synchronizer.cpp
LOC_CLIENT_SRC=LocalClient.cpp
DIST_CLIENT_SRC=DistantClient.cpp
MEDIATOR_SRC=Mediator.cpp
HEADERS=shdata.h
SYNCHRONIZER=Synchronizer
LOC_CLIENT=LocalClient
DIST_CLIENT=DistantClient
MEDIATOR=Mediator
all: $(SYNCHRONIZER) $(LOC_CLIENT) $(DIST_CLIENT) $(MEDIATOR)
$(SYNCHRONIZER): $(HEADERS) $(SYNCHRONIZER_SRC)
    $(CC) $(FLAGS) $(SYNCHRONIZER_SRC) -o $@
$(LOC_CLIENT): $(HEADERS) $(LOC_CLIENT_SRC)
    $(CC) $(FLAGS) $(LOC_CLIENT_SRC) -o $@
$(DIST_CLIENT): $(HEADERS) $(DIST_CLIENT_SRC)
    $(CC) $(FLAGS) $(DIST_CLIENT_SRC) -o $@
$(MEDIATOR): $(HEADERS) $(MEDIATOR_SRC)
    $(CC) $(FLAGS) $(MEDIATOR_SRC) -o $@
```

ShData.h

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <stdio.h>

#include <stdlib.h>
#include <iostream>

#define SEM_ID      2001      /* ключ массива семафоров */

#define SHM_ID_F    2005      /* ключ разделяемой памяти */
#define SHM_ID_S    2006      /* ключ разделяемой памяти */
#define SHM_ID_T    2007      /* ключ разделяемой памяти */

#define PERMS       0666      /* права доступа */

#define DATA_TYPE_SYNCHRONIZED 0 /* синхронизированные данные */
#define DATA_TYPE_FINISH 1 /* тип сообщения о том, что пора завершать обмен */
#define DATA_TYPE_CHANGED 2 /* данные изменины */

#define MAX_STRING   120

/* структура данных, помещаемого в разделяемую память */
typedef struct
{
//    int number;
    int type;
    char string [MAX_STRING];
} data_t;
```

Synchronizer.cpp

```
#include <stdio.h>
#include <string.h>
```

```

#include "shdata.h"

void sys_err (char *msg)
{
    std::cout << msg << std::endl;
    exit (1);
}

int main ()
{
    int semid;                                /* идентификатор семафора */
    int shmid_f, shmid_s, shmid_t;             /* идентификатор разделяемой памяти */
    data_t *data_p_f, *data_p_s, *data_p_t;      /* адрес данных в разделяемой памяти */
    char s[MAX_STRING];                        /* строка для передачи сообщения */

    /* создание массива семафоров из ТРЕХ элемента */
    if ((semid = semget (SEM_ID, 3, PERMS | IPC_CREAT)) < 0)
        sys_err ("server: can not create semaphore");
    std::cout << "Удалось создать семафор" << std::endl;

    /* создание нескольких сегментов разделяемой памяти */
    if ((shmid_f = shmget (SHM_ID_F, sizeof (data_t), PERMS | IPC_CREAT)) < 0)
        sys_err ("server: can not create shared memory segment 1");
    std::cout << "Удалось создать shared memory segment 1" << std::endl;

    if ((shmid_s = shmget (SHM_ID_S, sizeof (data_t), PERMS | IPC_CREAT)) < 0)
        sys_err ("server: can not create shared memory segment 2");
    std::cout << "Удалось создать shared memory segment 2" << std::endl;

    if ((shmid_t = shmget (SHM_ID_T, sizeof (data_t), PERMS | IPC_CREAT)) < 0)
        sys_err ("server: can not create shared memory segment 3");
    std::cout << "Удалось создать shared memory segment 3" << std::endl;

    /* подключение сегментов к адресному пространству процесса */
    if ((data_p_f = (data_t *) shmat (shmid_f, 0, 0)) == NULL)
        sys_err ("server: shared memory attach error");
    std::cout << "Удалось подключить 1 разделяемую память к процессу" << std::endl;

    if ((data_p_s = (data_t *) shmat (shmid_s, 0, 0)) == NULL)
        sys_err ("server: shared memory attach error");
    std::cout << "Удалось подключить 2 разделяемую память к процессу" << std::endl;

    if ((data_p_t = (data_t *) shmat (shmid_t, 0, 0)) == NULL)
        sys_err ("server: shared memory attach error");
    std::cout << "Удалось подключить 3 разделяемую память к процессу" << std::endl;

    semctl (semid, 0, SETVAL, 0);                /* установка семафора */
    semctl (semid, 1, SETVAL, 0);
    semctl (semid, 2, SETVAL, 0);
    data_p_f->type = DATA_TYPE_SYNCHRONIZED; // установили, что данные синхронизированы
    data_p_s->type = DATA_TYPE_SYNCHRONIZED;
    data_p_t->type = DATA_TYPE_SYNCHRONIZED;

    while (1)
    {
        if (data_p_f->type != DATA_TYPE_SYNCHRONIZED) // ПЕРВЫЙ СЕГМЕНТ ИЗМЕНЕН
        {
            if (semctl (semid, 0, GETVAL, 0))           /* блокировка - ждать */
                continue;

            semctl (semid, 0, SETVAL, 1);                /* установить блокировку на все! */
            semctl (semid, 1, SETVAL, 1);
            semctl (semid, 2, SETVAL, 1);

            /* обработка сообщения */
            if (data_p_f->type == DATA_TYPE_CHANGED)
            {
                std::cout << "Данные изменены в 1 сегменте разделяемой памяти: " << data_p_s->string <<
                " -> " << data_p_f->string << std::endl;
                strncpy (data_p_s->string, data_p_f->string, MAX_STRING); // синхронизируем
                strncpy (data_p_t->string, data_p_f->string, MAX_STRING);
                std::cout << "Все сегменты разделяемой памяти синхронизованы!" << std::endl;
            }
            if (data_p_f->type == DATA_TYPE_FINISH)
                break;
        }

        data_p_f->type = DATA_TYPE_SYNCHRONIZED; /* сообщение обработано */
        semctl (semid, 0, SETVAL, 0);             /* снять блокировку */
        semctl (semid, 1, SETVAL, 0);
    }
}

```

```

        semctl (semid, 2, SETVAL, 0);
    }

    if (data_p_s->type != DATA_TYPE_SYNCHRONIZED) // ВТОРОЙ СЕГМЕНТ ИЗМЕНЕН
    {
        if (semctl (semid, 1, GETVAL, 0)) /* блокировка - ждать */
            continue;

        semctl (semid, 0, SETVAL, 1); /* установить блокировку на все! */
        semctl (semid, 1, SETVAL, 1);
        semctl (semid, 2, SETVAL, 1);

        /* обработка сообщения */
        if (data_p_s->type == DATA_TYPE_CHANGED)
        {
            std::cout << "Данные изменены во 2 сегменте разделяемой памяти: " << data_p_f->string
<< " -> " << data_p_s->string << std::endl;
            strncpy (data_p_f->string, data_p_s->string, MAX_STRING); // синхронизируем
            strncpy (data_p_t->string, data_p_s->string, MAX_STRING);
            std::cout << "Все сегменты разделяемой памяти синхронизованы!" << std::endl;
        }
        if (data_p_s->type == DATA_TYPE_FINISH)
            break;

        data_p_s->type = DATA_TYPE_SYNCHRONIZED; /* сообщение обработано */
        semctl (semid, 0, SETVAL, 0); /* снять блокировку */
        semctl (semid, 1, SETVAL, 0);
        semctl (semid, 2, SETVAL, 0);
    }

    if (data_p_t->type != DATA_TYPE_SYNCHRONIZED) // ТРЕТИЙ СЕГМЕНТ ИЗМЕНЕН
    {
        if (semctl (semid, 2, GETVAL, 0)) /* блокировка - ждать */
            continue;

        semctl (semid, 0, SETVAL, 1); /* установить блокировку на все! */
        semctl (semid, 1, SETVAL, 1);
        semctl (semid, 2, SETVAL, 1);

        /* обработка сообщения */
        if (data_p_t->type == DATA_TYPE_CHANGED)
        {
            std::cout << "Данные изменены в 3 сегменте разделяемой памяти: " << data_p_s->string <<
" -> " << data_p_t->string << std::endl;
            strncpy (data_p_s->string, data_p_t->string, MAX_STRING); // синхронизируем
            strncpy (data_p_f->string, data_p_t->string, MAX_STRING);
            std::cout << "Все сегменты разделяемой памяти синхронизованы!" << std::endl;
        }
        if (data_p_t->type == DATA_TYPE_FINISH)
            break;

        data_p_t->type = DATA_TYPE_SYNCHRONIZED; /* сообщение обработано */
        semctl (semid, 0, SETVAL, 0); /* снять блокировку */
        semctl (semid, 1, SETVAL, 0);
        semctl (semid, 2, SETVAL, 0);
    }

    /* удаление массива семафоров */
    if (semctl (semid, 2, IPC_RMID, (struct semid_ds *) 0) < 0)
        sys_err ("server: semaphore 2 remove error");
    if (semctl (semid, 1, IPC_RMID, (struct semid_ds *) 0) < 0)
        sys_err ("server: semaphore 1 remove error");
    if (semctl (semid, 0, IPC_RMID, (struct semid_ds *) 0) < 0)
        sys_err ("server: semaphore 0 remove error");

    /* удаление сегмента разделяемой памяти */
    shmdt (data_p_f);
    if (shmctl (shmid_f, IPC_RMID, (struct shmid_ds *) 0) < 0)
        sys_err ("server: 1 shared memory remove error");
    shmdt (data_p_s);
    if (shmctl (shmid_s, IPC_RMID, (struct shmid_ds *) 0) < 0)
        sys_err ("server: 2 shared memory remove error");
    shmdt (data_p_t);
    if (shmctl (shmid_t, IPC_RMID, (struct shmid_ds *) 0) < 0)
        sys_err ("server: 3 shared memory remove error");

    std::cout << "BYE" << std::endl;
    exit (0);
}

```

LocalClient.cpp

```
#include <stdio.h>
#include <string.h>
#include "shdata.h"

void sys_err (char *msg)
{
    std::cout << msg << std::endl;
    exit (1);
}

int main ()
{
    int semid;                                /* идентификатор семафора */
    int shmid_f, shmid_s, shmid_t;             /* идентификатор разделяемой памяти */
    data_t *data_p_f, *data_p_s, *data_p_t;      /* адрес данных в разделяемой памяти */
    char s[MAX_STRING];

/* получение доступа к массиву из ТРЕХ семафоров */
if ((semid = semget (SEM_ID, 3, 0)) < 0)
    sys_err ("client: can not get semaphore");
std::cout << "Удалось подключиться к семафору" << std::endl;

/* получение доступа к сегменту разделяемой памяти */
if ((shmid_f = shmget (SHM_ID_F, sizeof (data_t), 0)) < 0)
    sys_err ("client: can not get shared memory segment 1");
std::cout << "Удалось подключиться к сегменту разделяемой памяти 1" << std::endl;

if ((shmid_s = shmget (SHM_ID_S, sizeof (data_t), 0)) < 0)
    sys_err ("client: can not get shared memory segment 2");
std::cout << "Удалось подключиться к сегменту разделяемой памяти 2" << std::endl;

if ((shmid_t = shmget (SHM_ID_T, sizeof (data_t), 0)) < 0)
    sys_err ("client: can not get shared memory segment 3");
std::cout << "Удалось подключиться к сегменту разделяемой памяти 3" << std::endl;

/* получение адреса сегмента */
if ((data_p_f = (data_t *) shmat (shmid_f, 0, 0)) == NULL)
    sys_err ("server: shared memory attach error");
std::cout << "Удалось получить адрес 1 сегмента разделяемой памяти" << std::endl;

if ((data_p_s = (data_t *) shmat (shmid_s, 0, 0)) == NULL)
    sys_err ("server: shared memory attach error");
std::cout << "Удалось получить адрес 2 сегмента разделяемой памяти" << std::endl;

if ((data_p_t = (data_t *) shmat (shmid_t, 0, 0)) == NULL)
    sys_err ("server: shared memory attach error");
std::cout << "Удалось получить адрес 3 сегмента разделяемой памяти" << std::endl;

int num = 0;
bool bKeepGoing = true;

char ex[120] = "exit";
char re[120] = "read";

while (bKeepGoing)
{
    if (num==0)
    {
        std::cout << "Введите номер сегмента разделяемой памяти (1-3) -> ";
        std::cin >> num;
        std::cout << std::endl << "Введите read, чтобы прочитать данные" << std::endl << "Введите exit, чтобы выйти" << std::endl << "Либо введите новые данные для перезаписи!" << std::endl;
        std::cout << " -> ";
        scanf ("%s", s);
    }

    while (semctl (semid, 0, GETVAL, 0) || data_p_f->type != DATA_TYPE_SYNCHRONIZED || semctl
(semid, 1, GETVAL, 0) || data_p_s->type != DATA_TYPE_SYNCHRONIZED || semctl (semid, 2, GETVAL, 0) ||
data_p_t->type != DATA_TYPE_SYNCHRONIZED)
    /*
     *   если данные не синхронизированы или сегмент блокирован - ждать
     */

    semctl (semid, 0, SETVAL, 1);
    semctl (semid, 1, SETVAL, 1);
    semctl (semid, 2, SETVAL, 1);      /* блокировать */
```

```

        switch (num)
    {
        case 1:
            if (strcmp(s,ex)!=0)
            {
                if (strcmp(s,re)==0)
                    std::cout << "Данные в 1 сегменте разделяемой памяти: " << data_p_f->string <<
std::endl;
                else
                {
                    /* записываем данные */
                    data_p_f->type = DATA_TYPE_CHANGED;
                    strncpy (data_p_f->string, s, MAX_STRING);
                }
            }
            else
            {
                /* ставим тип "завершение работы" */
                data_p_f->type = DATA_TYPE_FINISH;
                bKeepGoing = false;
            };
            num = 0;
            break;
        case 2:
            if (strcmp(s,ex)!=0)
            {
                if (strcmp(s,re)==0)
                    std::cout << "Данные во 2 сегменте разделяемой памяти: " << data_p_s->string <<
std::endl;
                else
                {
                    data_p_s->type = DATA_TYPE_CHANGED;
                    strncpy (data_p_s->string, s, MAX_STRING);
                }
            }
            else
            {
                data_p_s->type = DATA_TYPE_FINISH;
                bKeepGoing = false;
            };
            num = 0;
            break;
        case 3:
            if (strcmp(s,ex)!=0)
            {
                if (strcmp(s,re)==0)
                    std::cout << "Данные в 3 сегменте разделяемой памяти: " << data_p_t->string <<
std::endl;
                else
                {
                    data_p_t->type = DATA_TYPE_CHANGED;
                    strncpy (data_p_t->string, s, MAX_STRING);
                }
            }
            else
            {
                data_p_t->type = DATA_TYPE_FINISH;
                bKeepGoing = false;
            };
            num = 0;
            break;
        default:
            std::cout << "НЕВЕРНЫЙ номер сегмента разделяемой памяти" << std::endl;;
            num = 0;
    }
    semctl (semid, 0, SETVAL, 0);      /* отменить блокировку */
    semctl (semid, 1, SETVAL, 0);
    semctl (semid, 2, SETVAL, 0);
}
shmdt (data_p_f);                  /* отсоединить сегмент разделяемой памяти */
shmdt (data_p_s);
shmdt (data_p_t);
std::cout << "BYE" << std::endl;
exit (0);

}

```

Mediator.cpp

```

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

#include "shdata.h"

void sys_err (char *msg)
{
    std::cout << msg << std::endl;
    exit (1);
}

int main(int argc, char *argv[])
{
    // ----- здесь начинается часть клиента разделяемой памяти -----

    int semid;                                /* идентификатор семафора */
    int shmid_f, shmid_s, shmid_t;             /* идентификатор разделяемой памяти */
    data_t *data_p_f, *data_p_s, *data_p_t;      /* адрес данных в разделяемой памяти */
    char s[MAX_STRING];

    /* получение доступа к массиву из ТРЕХ семафоров */
    if ((semid = semget (SEM_ID, 3, 0)) < 0)
        sys_err ("client: can not get semaphore");
    std::cout << "Удалось подключиться к семафору" << std::endl;

    /* получение доступа к сегменту разделяемой памяти */
    if ((shmid_f = shmget (SHM_ID_F, sizeof (data_t), 0)) < 0)
        sys_err ("client: can not get shared memory segment 1");
    std::cout << "Удалось подключиться к сегменту разделяемой памяти 1" << std::endl;

    if ((shmid_s = shmget (SHM_ID_S, sizeof (data_t), 0)) < 0)
        sys_err ("client: can not get shared memory segment 2");
    std::cout << "Удалось подключиться к сегменту разделяемой памяти 2" << std::endl;

    if ((shmid_t = shmget (SHM_ID_T, sizeof (data_t), 0)) < 0)
        sys_err ("client: can not get shared memory segment 3");
    std::cout << "Удалось подключиться к сегменту разделяемой памяти 3" << std::endl;

    /* получение адреса сегмента */
    if ((data_p_f = (data_t *) shmat (shmid_f, 0, 0)) == NULL)
        sys_err ("server: shared memory attach error");
    std::cout << "Удалось получить адрес 1 сегмента разделяемой памяти" << std::endl;

    if ((data_p_s = (data_t *) shmat (shmid_s, 0, 0)) == NULL)
        sys_err ("server: shared memory attach error");
    std::cout << "Удалось получить адрес 2 сегмента разделяемой памяти" << std::endl;

    if ((data_p_t = (data_t *) shmat (shmid_t, 0, 0)) == NULL)
        sys_err ("server: shared memory attach error");
    std::cout << "Удалось получить адрес 3 сегмента разделяемой памяти" << std::endl;

    char ex[120] = "exit";
    char re[120] = "read";

    // ----

    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    listen(listenfd, 10);

    std::cout << "Слушаем порт 5000" << std::endl;

    while(1)

```

```

{
    std::cout << "слушаю!" << std::endl;
    connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);

    std::cout << "новое подключение!" << std::endl;

    std::cout << "читаем!" << std::endl;

    char sizeStr[4];
    read(connfd, sizeStr, 4);
    char recvBuff[atoi(sizeStr)];
    read(connfd, recvBuff, atoi(sizeStr));

    std::cout << "прочитали: " << recvBuff << std::endl;

    if (atoi(sizeStr) >= 120)
    {
        std::cout << "Строка слишком большая для сегмента разделяемой памяти" << std::endl;
        break;
    }

    char s[atoi(sizeStr)-1];
    for (int i = 0; i < atoi(sizeStr)-1; i++)
        s[i] = recvBuff[i+1];

    // ----- здесь начинается часть клиента разделяемой памяти -----

    while (semctl (semid, 0, GETVAL, 0) || data_p_f->type != DATA_TYPE_SYNCHRONIZED || semctl
(semid, 1, GETVAL, 0) || data_p_s->type != DATA_TYPE_SYNCHRONIZED || semctl (semid, 2, GETVAL, 0) || data_p_t->type != DATA_TYPE_SYNCHRONIZED)
    /*
     *   если данные не синхронизированы или сегмент блокирован - ждать
     */
    semctl (semid, 0, SETVAL, 1);
    semctl (semid, 1, SETVAL, 1);
    semctl (semid, 2, SETVAL, 1);      /* блокировать */

    switch (recvBuff[0])
    {
        case '1':
            if (strcmp(s,ex)!=0)
            {
                if (strcmp(s,re)==0)
                {
                    //
                    int size = strlen(data_p_f->string);
                    char sendBuff2[1024];
                    if (size>999)
                        sprintf(sendBuff2, "%d", size);
                    if (size>99 && size<1000)
                        sprintf(sendBuff2, "%c%d", '0', size);
                    if (size>9 && size<100)
                        sprintf(sendBuff2, "%c%c%d", '0', '0', size);
                    if (size<10)
                        sprintf(sendBuff2, "%c%c%c%d", '0', '0', '0', size);
                    for (int i = 0; i < size; i++)
                        sendBuff2[i+4] = data_p_f->string[i];
                    //

                    std::cout << "отправляем: " << sendBuff2 << std::endl;
                    write(connfd, sendBuff2, strlen(sendBuff2));

                    std::cout << "отправили!" << std::endl;
                }
            }
            else
            {
                /* записываем данные */
                data_p_f->type = DATA_TYPE_CHANGED;
                strncpy (data_p_f->string, s, MAX_STRING);
            }
        }
        else
        {
            /* ставим тип "завершение работы" */
            data_p_f->type = DATA_TYPE_FINISH;
        };
        break;
    case '2':
        if (strcmp(s,ex)!=0)
        {
            if (strcmp(s,re)==0)

```

```

    {
        // 
        int size = strlen(data_p_s->string);
        char sendBuff2[1024];
        if (size>999)
            sprintf(sendBuff2, "%d", size);
        if (size>99 && size<1000)
            sprintf(sendBuff2, "%c%d", '0', size);
        if (size>9 && size<100)
            sprintf(sendBuff2, "%c%c%d", '0', '0', size);
        if (size<10)
            sprintf(sendBuff2, "%c%c%c%d", '0', '0', '0', size);
        for (int i = 0; i < size; i++)
            sendBuff2[i+4] = data_p_s->string[i];
        //

        std::cout << "отправляем: " << sendBuff2 << std::endl;
        write(connfd, sendBuff2, strlen(sendBuff2));

        std::cout << "отправили!" << std::endl;
    }
    else
    {
        data_p_s->type = DATA_TYPE_CHANGED;
        strncpy (data_p_s->string, s, MAX_STRING);
    }
}
else
{
    data_p_s->type = DATA_TYPE_FINISH;
};

break;
case '3':
if (strcmp(s,ex)!=0)
{
    if (strcmp(s,re)==0)
    {
        // 
        int size = strlen(data_p_t->string);
        char sendBuff2[1024];
        if (size>999)
            sprintf(sendBuff2, "%d", size);
        if (size>99 && size<1000)
            sprintf(sendBuff2, "%c%d", '0', size);
        if (size>9 && size<100)
            sprintf(sendBuff2, "%c%c%d", '0', '0', size);
        if (size<10)
            sprintf(sendBuff2, "%c%c%c%d", '0', '0', '0', size);
        for (int i = 0; i < size; i++)
            sendBuff2[i+4] = data_p_t->string[i];
        //

        std::cout << "отправляем: " << sendBuff2 << std::endl;
        write(connfd, sendBuff2, strlen(sendBuff2));

        std::cout << "отправили!" << std::endl;
    }
    else
    {
        data_p_t->type = DATA_TYPE_CHANGED;
        strncpy (data_p_t->string, s, MAX_STRING);
    }
}
else
{
    data_p_t->type = DATA_TYPE_FINISH;
};

break;
default:
    std::cout << "НЕВЕРНЫЙ номер сегмента разделяемой памяти" << std::endl;
}

semctl (semid, 0, SETVAL, 0);      /* отменить блокировку */
semctl (semid, 1, SETVAL, 0);
semctl (semid, 2, SETVAL, 0);

// -----
close(connfd);
printf("\n Закрыли сокет \n");
sleep(1);
}

```

```
}
```

DistantClient.cpp

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

#include "shdata.h"

int main(int argc, char *argv[])
{
    int sockfd = 0;
    char sendBuff[1024];
    memset(sendBuff, '0', sizeof(sendBuff));
    struct sockaddr_in serv_addr;

    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n", argv[0]);
        return 1;
    }

    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);

    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\n inet_pton error occurred\n");
        return 1;
    }

    if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) // соединяемся
    {
        printf("\n Error : Connect Failed \n");
        return 1;
    }

    std::cout << "подключились!" << std::endl;

    char s[120];
    std::cout << "Получилось подключиться к промежуточному серверу!" << std::endl << "Введите номер
сегмента разделяемой памяти (1-3) -> ";
    std::cin >> sendBuff[0];
    std::cout << std::endl << "Введите read, чтобы прочитать данные" << std::endl << "Введите exit,
чтобы выйти" << std::endl << "Либо введите новые данные для перезаписи!" << std::endl;
    std::cout << " -> ";
    scanf ("%s", s);
    for (int i = 0; i < sizeof(s); i++)
        sendBuff[i+1] = s[i];

    //
    int size = strlen(sendBuff);
    char sendBuff2[1024];
    if (size>999)
        sprintf(sendBuff2, "%d", size);
    if (size>99 && size<1000)
        sprintf(sendBuff2, "%c%d", '0', size);
    if (size>9 && size<100)
        sprintf(sendBuff2, "%c%c%d", '0', '0', size);
    if (size<10)
        sprintf(sendBuff2, "%c%c%c%d", '0', '0', '0', size);
    for (int i = 0; i < size; i++)
        sendBuff2[i+4] = sendBuff[i];
    //
```

```
    std::cout << "отправляем : " << sendBuff2 << std::endl;
    write(sockfd, sendBuff2, strlen(sendBuff2));
    std::cout << "отправили!" << std::endl;
    char re[120] = "read";
    if (strcmp(s,re)==0)
    {
        char sizeStr[4];
        read(sockfd, sizeStr, 4);
        char recvBuff[atoi(sizeStr)];
        read(sockfd, recvBuff, atoi(sizeStr));
        std::cout << "получили: " << recvBuff << std::endl;
    }
    return 0;
}
```