Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет "Высшая школа экономики"
Московский институт электроники и математики
им. А. Н. Тихонова НИУ ВШЭ
Департамент компьютерной инженерии

Лабораторная работа № 6
по теме:
«Разделяемая память и нити исполнения»

По дисциплине

«Операционные системы»

Группа: **БПМ 243**

Студент: **Николаев Степан Денисович**

Москва 2026

Цель работы

Научиться обмениваться данными через разделяемую память System V и познакомиться с потоками pthread. Написать писателя и читателя, а также вариант с двумя потоками внутри одного процесса.

Ход работы

Все программы собирал и запускал в виртуальной машине с Ubuntu в UTM под обычным пользователем marmos, компилятор gcc с ключами -Wall -Wextra, предупреждений нет. Права суперпользователя для shmget, shmat, shmdt и shmctl не понадобились: сегмент создаётся с правами 0666.

Писатель writer.c открывает собственный исходный файл через open, узнаёт размер вызовом lseek(fd, 0, SEEK_END) и создаёт сегмент через shmget с ключом 0x4F5306. Размер сегмента на байт больше файла: туда кладётся завершающий \0. Затем сегмент присоединяется через shmat, содержимое файла читается прямо в него вызовом read, после чего писатель отсоединяется через shmdt и завершается. Библиотечные функции для работы с файлами не использовал.

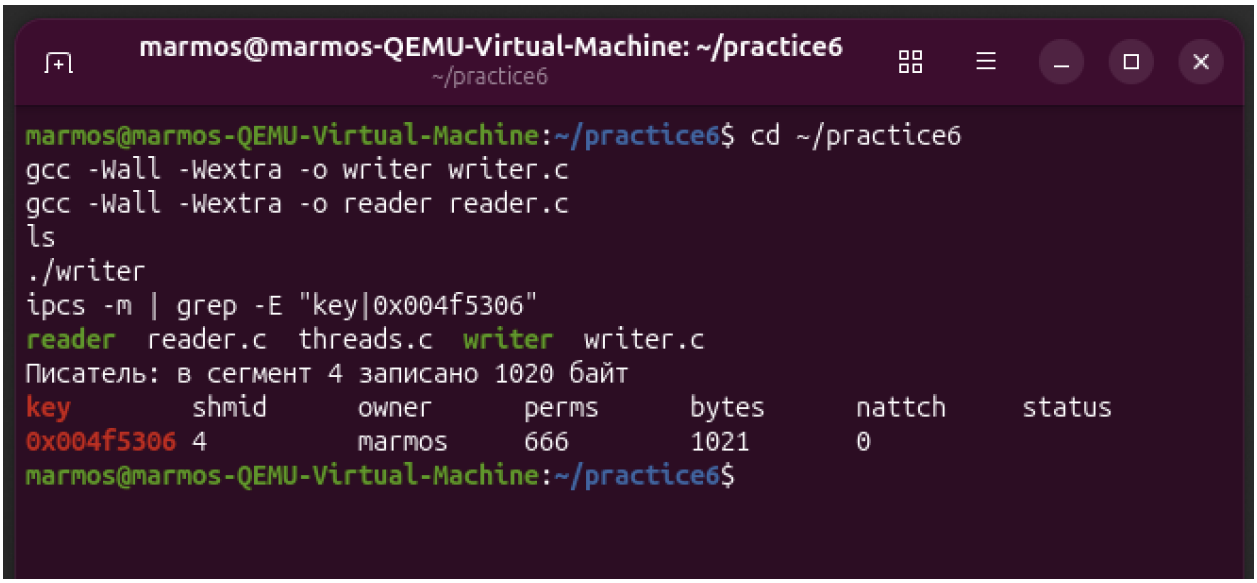

Рис. 1. Компиляция и запуск писателя, сегмент в ipcs -m.

Писатель записал 1020 байт в сегмент с идентификатором 4, а ipcs -m показывает размер 1021 байт (с завершающим \0). nattch равен 0, так как писатель уже завершился, но сегмент остался в системе.

Читатель reader.c находит сегмент по тому же ключу и узнаёт его размер через shmctl с командой IPC_STAT. Потом присоединяет сегмент (только для чтения), выводит текст через write, отсоединяется и удаляет сегмент вызовом shmctl с IPC_RMID.

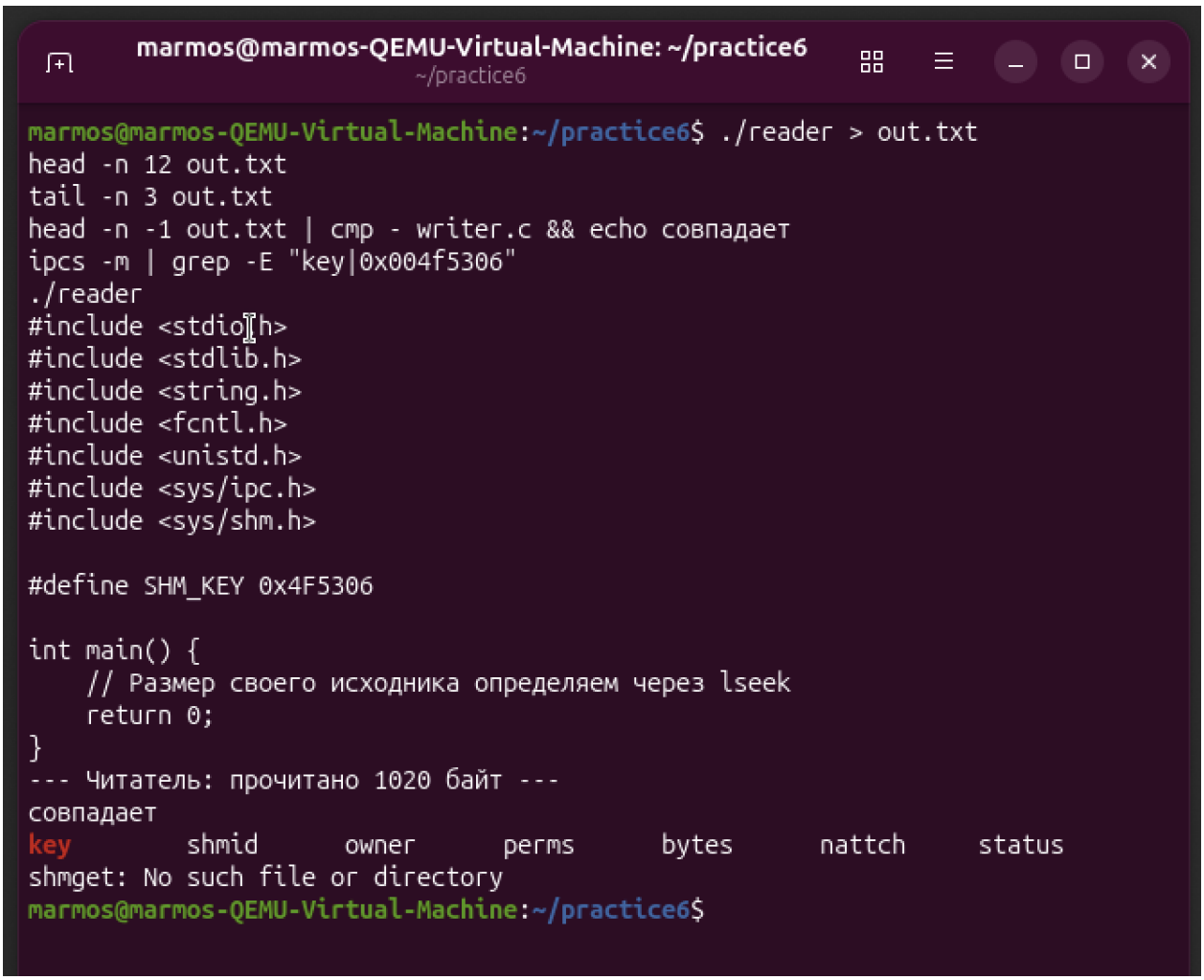

Рис. 2. Запуск читателя и проверка результата.

Вывод читателя сохранил в out.txt и сравнил с writer.c через cmp: текст совпал (последняя строка в файле — служебная от читателя, её отрезал через head -n -1). После запуска читателя ipcs -m показывает пустой список, а повторный запуск reader ожидаемо выдаёт «No such file or directory»: сегмента в системе уже нет.

Вариант с потоками

В threads.c создаются два потока pthread. Поток-писатель создаёт сегмент с ключом IPC_PRIVATE и записывает в него строку. Поток-читатель сначала ждёт завершения писателя через pthread_join, потом читает строку из сегмента и удаляет его. Без ожидания читатель мог бы начать раньше, чем писатель успеет что-то записать. Собирал с ключом -pthread.

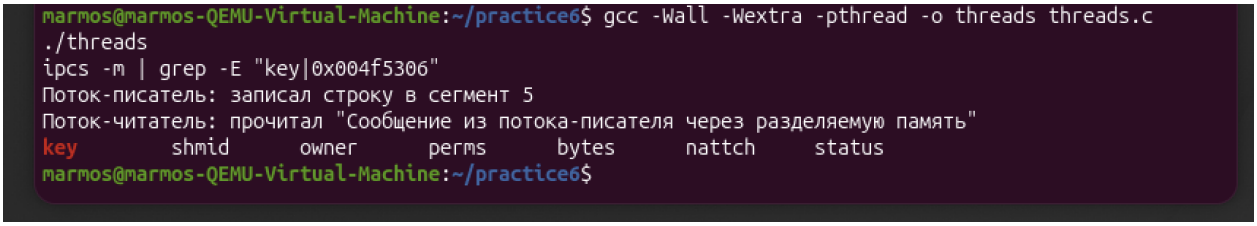

Рис. 3. Запуск программы с двумя потоками.

Поток-писатель записал строку в сегмент 5, поток-читатель её прочитал. После завершения программы ipcs -m пустой: сегмент удалён.

Вывод. Сделал обмен данными между двумя процессами через сегмент разделяемой памяти и вариант с двумя потоками. Разделяемая память быстрая, но сама по себе не синхронизирует доступ: в процессном варианте читатель надо запускать строго после писателя, а в потоковом пришлось добавить pthread_join. Синхронизацию семафорами сделаю в следующей работе.

Приложение writer.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_KEY 0x4F5306

int main() {
    // Размер своего исходника определяем через lseek
    int fd = open("writer.c", O_RDONLY);
    if (fd == -1) {
        perror("open");
        return 1;
    }
    off_t size = lseek(fd, 0, SEEK_END);
    lseek(fd, 0, SEEK_SET);

    // +1 байт под завершающий \0
    int shmid = shmget(SHM_KEY, size + 1, IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget");
        return 1;
    }
    char *mem = shmat(shmid, NULL, 0);
    if (mem == (void *)-1) {
        perror("shmat");
        return 1;
    }

    ssize_t n = read(fd, mem, size);
    if (n == -1) {
        perror("read");
        return 1;
    }
    mem[n] = '\0';
    close(fd);

    printf("Писатель: в сегмент %d записано %zd байт\n", shmid, n);
    shmdt(mem);
    return 0;
}
```

Приложение reader.c:

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

#define SHM_KEY 0x4F5306

int main() {
    int shmid = shmget(SHM_KEY, 0, 0666);
    if (shmid == -1) {
        perror("shmget");
        return 1;
    }

    // Размер сегмента берём из shmctl, чтобы знать, сколько печатать
    struct shmid_ds info;
    if (shmctl(shmid, IPC_STAT, &info) == -1) {
        perror("shmctl IPC_STAT");
        return 1;
    }

    char *mem = shmat(shmid, NULL, SHM_RDONLY);
    if (mem == (void *)-1) {
        perror("shmat");
        return 1;
    }

    write(STDOUT_FILENO, mem, info.shm_segsz - 1);
    printf("--- Читатель: прочитано %zu байт ---\n", (size_t)info.shm_segsz - 1);

    shmdt(mem);
    // Удаляем сегмент из системы
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
        perror("shmctl IPC_RMID");
        return 1;
    }
    return 0;
}
```

Приложение threads.c:

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 256

static int shmid;
static pthread_t writer_tid;

// Поток-писатель: создаёт сегмент и кладёт в него строку
static void *writer_thread(void *arg) {
    (void)arg;
    shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0600);
    if (shmid == -1) {
        perror("shmget");
        return NULL;
    }
    char *mem = shmat(shmid, NULL, 0);
    if (mem == (void *)-1) {
        perror("shmat");
        return NULL;
    }
    strcpy(mem, "Сообщение из потока-писателя через разделяемую память");
    printf("Поток-писатель: записал строку в сегмент %d\n", shmid);
    shmdt(mem);
    return NULL;
}

// Поток-читатель: ждёт писателя, читает строку и удаляет сегмент
static void *reader_thread(void *arg) {
    (void)arg;
    pthread_join(writer_tid, NULL);
    char *mem = shmat(shmid, NULL, SHM_RDONLY);
    if (mem == (void *)-1) {
        perror("shmat");
        return NULL;
    }
    printf("Поток-читатель: прочитал \"%s\"\n", mem);
    shmdt(mem);
    shmctl(shmid, IPC_RMID, NULL);
    return NULL;
}

int main() {
    pthread_t reader_tid;
    pthread_create(&writer_tid, NULL, writer_thread, NULL);
    pthread_create(&reader_tid, NULL, reader_thread, NULL);
    pthread_join(reader_tid, NULL);
    return 0;
}
```