Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-211Б-23

Студент: Леоненкова Е.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.12.24

Москва, 2024

Постановка задач

Вариант 18.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в mmaped_file_2 или в mmaped_file_2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: нечетные строки отправляются в mmaped_file_1, четные в mmaped_file_2.
Дочерние процессы удаляют все гласные из строк.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int execv(const char *__path, char *const *__argv); – заменяет образ текущего процесса на образ нового процесса, определённого в пути path.
- pid_t wait(int *__stat_loc); – используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.
- int shm_open(const char *name, int oflag, mode_t mode); – создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
- int shm_unlink(const char *name); – удаляется имя объекта разделяемой памяти и, как только все процессы завершили работу с объектом и отменили его распределение, очищают пространство и уничтожают связанную с ним область памяти.
- int ftruncate(int fd, off_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
- void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.

- int munmap(void *start, size_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти".
- sem_t *sem_open(const char *name, int oflag); ИЛИ sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – создаёт новый семафор или открывает уже существующий.
- int sem_wait(sem_t *sem); – уменьшает значение семафора на 1. Если семафор в данный момент имеет нулевое значение, то вызов блокируется до тех пор, пока либо не станет возможным выполнить уменьшение.
- int sem_post(sem_t *sem); – увеличивает значение семафора на 1.
- int sem_unlink(const char *name); – удаляет имя семафора из системы. После вызова этой функции другие процессы больше не смогут открыть этот семафор по имени.
- int sem_close(sem_t *sem); – закрывает указанный семафор, освобождая ресурсы, связанные с ним.

Программа parent.c получает на вход два аргумента – два имени файла для двух дочерних процессов. Далее создаются два файла для общей памяти, в которые будут записываться строки. Создаются два семафора для каждого дочернего процесса для синхронизации работы с общей памятью. Для каждого процесса с помощью fork() создается новый процесс.

После успешного создания, родитель запускает child.c, передавая ей параметры: имя файла, в который дочерний процесс будет записывать результат, и название общей памяти и семафоров, с которыми дочерний процесс будет работать. Родитель считывает строки с консоли, если нечетная, то отправляется в первую область общей памяти – mmaped_file_1, иначе – во вторую - mmaped_file_2.

В child.c получаются данные, открывается файл для записи, создается общая память для обмена строчками и подключаются семафоры. После получения строчки дочерний процесс удаляет из нее все гласны. После окончания ввода закрывает общую память и семафоры, ждем завершния дочерних процессов с помощью wait().

Код программы

Parent.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>

const int MAX_LENGTH = 255;
// доч процесс
int create_process()
{
    pid_t pid = fork();
    if (pid == -1)
    {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    return pid;
}

int main()
{
    const int number_processes = 2;
    const char *mmapped_file_names[2];
    mmapped_file_names[1] = "mmaped_file_1";
    mmapped_file_names[0] = "mmaped_file_2";
    const char *semaphores_names[2];
    semaphores_names[0] = "/semaphoreOne";
    semaphores_names[1] = "/semaphoreTwo";
    const char *semaphoresForParent_names[2];
    semaphoresForParent_names[0] = "/semaphoresForParentOne";
    semaphoresForParent_names[1] = "/semaphoresForParentTwo";

    // Считываем имена для дочерних процессов
```

```
    char file_names[number_processes][MAX_LENGTH];
    for (int i = 0; i < number_processes; ++i)
    {
        printf("Enter filename for child%d: ", i + 1);
        if (fgets(file_names[i], MAX_LENGTH, stdin) == NULL)
        {
            perror("fgets");
            exit(EXIT_FAILURE);
        }

        int str_len = strlen(file_names[i]);
        if (file_names[i][str_len - 1] == 10)
        { // убираем перенос строки
            file_names[i][str_len - 1] = 0;
        }
    }

    // Создаем memory-mapped файлы и получаем их дескрипторы
    int mmapped_file_descriptors[number_processes];
    char *mmapped_file_pointers[number_processes];
    for (int i = 0; i < number_processes; ++i)
    {
        mmapped_file_descriptors[i] = shm_open(mmapped_file_names[i], O_RDWR |
O_CREAT, 0777);
        if (mmapped_file_descriptors[i] == -1)
        {
            perror("shm_open");
            exit(EXIT_FAILURE);
        }

        if (ftruncate(mmapped_file_descriptors[i], MAX_LENGTH) == -1)
        {
            perror("ftruncate");
            exit(EXIT_FAILURE);
        }

        mmapped_file_pointers[i] = mmap(NULL, MAX_LENGTH, PROT_READ | PROT_WRITE,
MAP_SHARED, mmapped_file_descriptors[i], 0);
        if (mmapped_file_pointers[i] == MAP_FAILED)
        {
```

```
            perror("mmap");
            exit(EXIT_FAILURE);
        }
    }

    // создаем семафоры
    sem_t *semaphores[number_processes][2];
    for (int i = 0; i < number_processes; ++i)
    {
        semaphores[i][0] = sem_open(semaphores_names[i], O_CREAT, 0777, 0);
        if (semaphores[i][0] == SEM_FAILED)
        {
            perror("sem_open");
            exit(EXIT_FAILURE);
        }

        semaphores[i][1] = sem_open(semaphoresForParent_names[i], O_CREAT, 0777,
1);
        if (semaphores[i][1] == SEM_FAILED)
        {
            perror("sem_open");
            exit(EXIT_FAILURE);
        }
    }
    // создаем дочерние пр
    for (int index = 0; index < number_processes; ++index)
    {
        pid_t process_id = create_process();
        if (process_id == 0)
        {
            // Дочерний процесс
            execl("./child", "child", file_names[index], mmapped_file_names[index],
semaphores_names[index], semaphoresForParent_names[index], NULL);
            perror("exec");
            exit(EXIT_FAILURE);
        }
    }

    // Считываем вводные данные из консоли
    char string[MAX_LENGTH];
```

```
    int counter = 0;
    while (fgets(string, MAX_LENGTH, stdin))
    {
        if (++counter % 2 == 1)
        {
            sem_wait(semaphores[0][1]);

            strcpy(mmapped_file_pointers[0], string);

            sem_post(semaphores[0][0]);
        }
        else
        {
            sem_wait(semaphores[1][1]);

            strcpy(mmapped_file_pointers[1], string);

            sem_post(semaphores[1][0]);
        }
    }

    sem_wait(semaphores[0][1]);
    sem_wait(semaphores[1][1]);
    mmapped_file_pointers[0][0] = 0;
    mmapped_file_pointers[1][0] = 0;
    sem_post(semaphores[0][0]);
    sem_post(semaphores[1][0]);

    wait(NULL);
    // освобождаем
    for (int i = 0; i < number_processes; ++i)
    {
        munmap(mmapped_file_pointers[i], MAX_LENGTH);
        shm_unlink(mmapped_file_names[i]);
        close(mmapped_file_descriptors[i]);
        sem_close(semaphores[i][0]);
        sem_close(semaphores[i][1]);
        sem_unlink(semaphores_names[i]);
        sem_unlink(semaphoresForParent_names[i]);
```

```
    }

    return 0;
}
```

Child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>

const int MAX_LENGTH = 255;

int main(int argc, char* argv[]) {
    char file_name[MAX_LENGTH];
    char mmapped_file_name[MAX_LENGTH];
    strcpy(file_name, argv[1]);
    strcpy(mmapped_file_name, argv[2]);

    char semaphore_name[MAX_LENGTH];
    strcpy(semaphore_name, argv[3]);
    char semaphoreForParent_name[MAX_LENGTH];
    strcpy(semaphoreForParent_name, argv[4]);

    int file_descriptor = open(file_name, O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
    if (file_descriptor == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    int mmapped_file_descriptor = shm_open(mmapped_file_name, O_RDWR, 0777);
    if (mmapped_file_descriptor == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }
```

```
    char* mmapped_file_pointer = mmap(NULL, MAX_LENGTH, PROT_READ | PROT_WRITE,
MAP_SHARED, mmapped_file_descriptor, 0);
    if (mmapped_file_pointer == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    sem_t* semaphore = sem_open(semaphore_name, 0);
    if (semaphore == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

    sem_t* semaphoreForParent = sem_open(semaphoreForParent_name, 0);
    if (semaphoreForParent == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

    char vowels[] = {'a', 'e', 'i', 'o', 'u', 'y', 'A', 'E', 'I', 'O', 'U', 'Y'};
    char string[MAX_LENGTH];
    while (1) {
        sem_wait(semaphore);

        strcpy(string, mmapped_file_pointer);

        sem_post(semaphoreForParent);

        if (strlen(string) == 0) {
            break;
        }

        for (int index = 0; index < strlen(string); ++index) {
            if (memchr(vowels, string[index], sizeof(vowels)) == NULL) {
                write(file_descriptor, &string[index], 1);
            }
        }
    }

    munmap(mmapped_file_pointer, MAX_LENGTH);
```

```
    sem_close(semaphore);
    sem_close(semaphoreForParent);
    close(file_descriptor);

    return 0;
}
```

Протокол работы программы

```
leoelena@DESKTOP-HJEL67G:/mnt/c/Users/Елена/Desktop/OS_lab/OS/lab_3/sr
c$ ./parent
Enter filename for child1: file_1.txt
Enter filename for child2: file_2.txt
Hello world
My name is Lena
I'm soooo tyred
Veeeeryyy
Вывод в file_1.txt:
Hll wrld
'm s trd
вывод в file_2.txt:
M nm s Ln
vr
leoelena@DESKTOP-HJEL67G:/mnt/c/Users/Елена/Desktop/OS_lab/OS/lab_3/sr
c$ strace ./parent
execve("./parent", ["./parent"], 0x7ffef80e7190 /* 36 vars */) = 0
brk(NULL)                               = 0x559064fe4000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f66a7b14000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=20115, ...}) = 0
mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f66a7b0f000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f66a78fd000
mmap(0x7f66a7925000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000)
= 0x7f66a7925000
mmap(0x7f66a7aad000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) =
0x7f66a7aad000
mmap(0x7f66a7afc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000)
= 0x7f66a7afc000
mmap(0x7f66a7b02000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7f66a7b02000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f66a78fa000
arch_prctl(ARCH_SET_FS, 0x7f66a78fa740) = 0
set_tid_address(0x7f66a78faa10)         = 5851
set_robust_list(0x7f66a78faa20, 24)     = 0
rseq(0x7f66a78fb060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f66a7afc000, 16384, PROT_READ) = 0
mprotect(0x559063a5c000, 4096, PROT_READ) = 0
mprotect(0x7f66a7b4c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f66a7b0f000, 20115)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x5), ...}) = 0
getrandom("\xaf\x72\x31\xf8\xa9\x8c\xd4\x66", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x559064fe4000
brk(0x559065005000)                     = 0x559065005000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x5), ...}) = 0
write(1, "Enter filename for child1: ", 27Enter filename for child1: ) = 27
read(0, output_1.txt
"output_1.txt\n", 1024)         = 13
write(1, "Enter filename for child2: ", 27Enter filename for child2: ) = 27
read(0, output_2.txt
"output_2.txt\n", 1024)         = 13
openat(AT_FDCWD, "/dev/shm/mmaped_file_2", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0777) = 3
ftruncate(3, 255)                       = 0
mmap(NULL, 255, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f66a7b13000
```

```
openat(AT_FDCWD, "/dev/shm/mmaped_file_1", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0777) = 4
ftruncate(4, 255)                       = 0
mmap(NULL, 255, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66a7b12000
openat(AT_FDCWD, "/dev/shm/sem.semaphoreOne", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file
or directory)
getrandom("\x54\xf3\x4c\xae\x34\x9c\x73\xd1", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.G2klpL", 0x7ffd1d2abc80, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.G2klpL", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0777) = 5
write(5, "\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f66a7b11000
link("/dev/shm/sem.G2klpL", "/dev/shm/sem.semaphoreOne") = 0
fstat(5, {st_mode=S_IFREG|0755, st_size=32, ...}) = 0
unlink("/dev/shm/sem.G2klpL")           = 0
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.semaphoresForParentOne", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No
such file or directory)
getrandom("\xdd\xc2\xe6\x02\x4d\xb0\x23\x2f", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.xz2yWN", 0x7ffd1d2abc80, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.xz2yWN", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0777) = 5
write(5, "\1\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f66a7b10000
link("/dev/shm/sem.xz2yWN", "/dev/shm/sem.semaphoresForParentOne") = 0
fstat(5, {st_mode=S_IFREG|0755, st_size=32, ...}) = 0
unlink("/dev/shm/sem.xz2yWN")           = 0
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.semaphoreTwo", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file
or directory)
getrandom("\x35\x43\xdd\x2d\x27\x20\x68\x4e", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.n04aDz", 0x7ffd1d2abc80, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.n04aDz", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0777) = 5
write(5, "\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f66a7b0f000
link("/dev/shm/sem.n04aDz", "/dev/shm/sem.semaphoreTwo") = 0
fstat(5, {st_mode=S_IFREG|0755, st_size=32, ...}) = 0
unlink("/dev/shm/sem.n04aDz")           = 0
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.semaphoresForParentTwo", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No
such file or directory)
getrandom("\x5a\xde\xd6\x12\x8f\x01\xcb\x0c", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.8R9UX5", 0x7ffd1d2abc80, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.8R9UX5", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0777) = 5
write(5, "\1\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f66a78f9000
link("/dev/shm/sem.8R9UX5", "/dev/shm/sem.semaphoresForParentTwo") = 0
fstat(5, {st_mode=S_IFREG|0755, st_size=32, ...}) = 0
unlink("/dev/shm/sem.8R9UX5")           = 0
close(5)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f66a78faa10) = 6156
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f66a78faa10) = 6157
read(0, Hello world
"Hello world\n", 1024)          = 12
futex(0x7f66a7b11000, FUTEX_WAKE, 1)    = 1
read(0, Doing homework
"Doing homework\n", 1024)       = 15
futex(0x7f66a7b0f000, FUTEX_WAKE, 1)    = 1
read(0, first
"first\n", 1024)                = 6
futex(0x7f66a7b11000, FUTEX_WAKE, 1)    = 1
read(0, second
"second\n", 1024)               = 7
futex(0x7f66a7b0f000, FUTEX_WAKE, 1)    = 1
```

```
read(0, dog frog
"dog frog\n", 1024)             = 9
futex(0x7f66a7b11000, FUTEX_WAKE, 1)    = 1
read(0, cat
"cat \n", 1024)                 = 5
futex(0x7f66a7b0f000, FUTEX_WAKE, 1)    = 1
read(0, "", 1024)                       = 0
futex(0x7f66a7b11000, FUTEX_WAKE, 1)    = 1
futex(0x7f66a7b0f000, FUTEX_WAKE, 1)    = 1
wait4(-1, NULL, 0, NULL)                = 6157
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6157, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
munmap(0x7f66a7b13000, 255)             = 0
unlink("/dev/shm/mmaped_file_2")        = 0
close(3)                                = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6156, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
munmap(0x7f66a7b11000, 32)              = 0
munmap(0x7f66a7b10000, 32)              = 0
unlink("/dev/shm/sem.semaphoreOne")     = 0
unlink("/dev/shm/sem.semaphoresForParentOne") = 0
munmap(0x7f66a7b12000, 255)             = 0
unlink("/dev/shm/mmaped_file_1")        = 0
close(4)                                = 0
munmap(0x7f66a7b0f000, 32)              = 0
munmap(0x7f66a78f9000, 32)              = 0
unlink("/dev/shm/sem.semaphoreTwo")     = 0
unlink("/dev/shm/sem.semaphoresForParentTwo") = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В процессе выполнения данной лабораторной работы я изучила новые системные вызовы на языке Си, которые позволяют эффективно работать с разделяемой памятью и семафорами. Освоила передачу данных между процессами через shared memory и управление доступом с использованием семафоров. Затруднений в ходе выполнения лабораторной работы не возникло, все задачи удалось успешно реализовать.