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

**Вариант 18.**

# Постановка задач

**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:  
  
#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\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", 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\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", 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\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", 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\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", 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.