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-214Б-23

Студент: Миронов Д. А.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 12.12.24

Москва, 2024

**Постановка задачи**

**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 дочернему процессу 1 или в 2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод. Правило фильтрации: Правило фильтрации: нечетные строки отправляются в процесс 1, четные в 2. Дочерние процессы удаляют все гласные из строк.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* shm\_open() – создание разделяемой памяти.
* mmap() – отображение дескриптора разделяемой памяти в память.
* sem\_open() – создает семафор.
* fork() – создает дочерний процесс.
* sem\_post() – увеличивает значение семафора, освобождая ресурс.
* wait() – блокирует родительский процесс, пока дочерний существует.
* munmap() – освобождение памяти, выделенной в mmap.
* shm\_unlink() – удаляет объект разделяемой памяти.
* sem\_close() – закрывает дескриптор семафора.
* sem\_unlink() – удаление семафора.

**Код программы**

**server.c**

#include <stdint.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <stdio.h>

#define BUFFER\_SIZE 1024

#define SHM\_NAME "/my\_shared\_memory"

#define SEM\_NAME\_CHILD1 "/my\_semaphore\_child1"

#define SEM\_NAME\_CHILD2 "/my\_semaphore\_child2"

**static** **char** CLIENT1\_PROGRAM\_NAME[] = "client1";

**static** **char** CLIENT2\_PROGRAM\_NAME[] = "client2";

**int** main(**int** argc, **char** \*\*argv) {

**if** (argc == 2) {

**char** msg[1024];

uint32\_t len = snprintf(msg, **sizeof**(msg) - 1, "usage: %s filename\n", argv[0]);

write(STDERR\_FILENO, msg, len);

exit(EXIT\_SUCCESS);

}

**char** progpath[1024];

{

// NOTE: Read full program path, including its name

ssize\_t len = readlink("/proc/self/exe", progpath, **sizeof**(progpath) - 1);

**if** (len == -1) {

**const** **char** msg[] = "error: failed to read full program path\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

**while** (progpath[len] != '/')

--len;

progpath[len] = '\0';

}

**char** file1\_path[1024];

{

**char** filename[128];

read(fileno(stdin), filename, **sizeof**(filename));

filename[strcspn(filename, "\n")] = '\0';

snprintf(file1\_path, **sizeof**(file1\_path) - 1, "%s/%s", progpath, filename);

}

**char** file2\_path[1024];

{

**char** filename[128];

read(fileno(stdin), filename, **sizeof**(filename));

filename[strcspn(filename, "\n")] = '\0';

snprintf(file2\_path, **sizeof**(file2\_path) - 1, "%s/%s", progpath, filename);

}

// Создание разделяемой памяти

**int** shm\_fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);

**if** (shm\_fd == -1) {

**const** **char** msg[] = "Не удалось создать разделяемую память\n";

write(STDOUT\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

ftruncate(shm\_fd, BUFFER\_SIZE);

// Отображение разделяемой памяти

**char** \*shared\_memory = (**char** \*)mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

**if** (shared\_memory == MAP\_FAILED) {

**const** **char** msg[] = "Не удалось отобразить разделяемую память\n";

write(STDOUT\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

// Создание семафоров

sem\_t \*sem\_child1 = sem\_open(SEM\_NAME\_CHILD1, O\_CREAT, 0666, 0);

sem\_t \*sem\_child2 = sem\_open(SEM\_NAME\_CHILD2, O\_CREAT, 0666, 0);

**if** (sem\_child1 == SEM\_FAILED || sem\_child2 == SEM\_FAILED) {

**const** **char** msg[] = "Не удалось создать семафоры\n";

write(STDOUT\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

**const** pid\_t child = fork();

**switch** (child) {

**case** -1: {

**const** **char** msg[] = "error: failed to spawn new process\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

} **break**;

**case** 0: { // NOTE: We're a child, child doesn't know its pid after fork

pid\_t pid = getpid(); // NOTE: Get child PID

{

**char** msg[64];

**const** int32\_t length = snprintf(msg, **sizeof**(msg),"%d: I'm a child\n", pid);

write(STDOUT\_FILENO, msg, length);

}

{

**char** path[1024];

snprintf(path, **sizeof**(path) - 1, "%s/%s", progpath, CLIENT1\_PROGRAM\_NAME);

**char** \***const** args[] = {CLIENT1\_PROGRAM\_NAME, file1\_path, NULL};

int32\_t status = execv(path, args);

**if** (status == -1) {

**const** **char** msg[] = "error: failed to exec into new exectuable image\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

}

} **break**;

**default**: { // NOTE: We're a parent, parent knows PID of child after fork

**const** pid\_t child2 = fork();

**switch** (child2) {

**case** -1: {

**const** **char** msg[] = "error: failed to spawn new process\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

} **break**;

**case** 0: { // NOTE: We're a child, child doesn't know its pid after fork

pid\_t pid2 = getpid(); // NOTE: Get child PID

{

**char** msg[64];

**const** int32\_t length = snprintf(msg, **sizeof**(msg),"%d: I'm a child\n", pid2);

write(STDOUT\_FILENO, msg, length);

}

{

**char** path[1024];

snprintf(path, **sizeof**(path) - 1, "%s/%s", progpath, CLIENT2\_PROGRAM\_NAME);

**char** \***const** args[] = {CLIENT2\_PROGRAM\_NAME, file2\_path, NULL};

int32\_t status = execv(path, args);

**if** (status == -1) {

**const** **char** msg[] = "error: failed to exec into new exectuable image\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

}

} **break**;

**default**: { // NOTE: We're a parent, parent knows PID of child after fork

**int** bytes, flag = 1;

**char** buf[1024];

**while** ((bytes = read(STDIN\_FILENO, buf, **sizeof**(buf) - 1)) > 0) {

buf[strcspn(buf, "\n")] = '\0';

strncpy(shared\_memory, buf, bytes);

**if** (flag) {

sem\_post(sem\_child1);

flag = 0;

} **else** {

sem\_post(sem\_child2);

flag = 1;

}

}

strncpy(shared\_memory, "", BUFFER\_SIZE);

sem\_post(sem\_child1);

sem\_post(sem\_child2);

// NOTE: `wait` blocks the parent until child exits

**int** child\_status;

wait(&child\_status);

**if** (child\_status != EXIT\_SUCCESS) {

**const** **char** msg[] = "error: child exited with error\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(child\_status);

}

munmap(shared\_memory, BUFFER\_SIZE);

shm\_unlink(SHM\_NAME);

sem\_close(sem\_child1);

sem\_close(sem\_child2);

sem\_unlink(SEM\_NAME\_CHILD1);

sem\_unlink(SEM\_NAME\_CHILD2);

} **break**;

}

} **break**;

}

}

**client1.c**

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#define BUFFER\_SIZE 1024

#define SHM\_NAME "/my\_shared\_memory"

#define SEM\_NAME\_CHILD1 "/my\_semaphore\_child1"

**int** main(**int** argc, **char** \*\*argv) {

**const** **char** \*vowels = "aeiouAEIOU";

printf("<<%s>>\n", argv[1]);

int32\_t file = open(argv[1], O\_WRONLY | O\_CREAT | O\_APPEND, 0600);

**if** (file == -1) {

**const** **char** msg[] = "error: failed to open requested file\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

// Открытие разделяемой памяти

**int** shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);

**if** (shm\_fd == -1) {

write(STDOUT\_FILENO, "Не удалось открыть разделяемую память", **sizeof**("Не удалось открыть разделяемую память"));

exit(EXIT\_FAILURE);

}

// Отображение разделяемой памяти

**char** \*shared\_memory = (**char** \*)mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

**if** (shared\_memory == MAP\_FAILED) {

write(STDOUT\_FILENO, "Не удалось отобразить разделяемую память", **sizeof**("Не удалось отобразить разделяемую память"));

exit(EXIT\_FAILURE);

}

// Открытие семафоров

sem\_t \*sem\_child1 = sem\_open(SEM\_NAME\_CHILD1, 0);

**if** (sem\_child1 == SEM\_FAILED) {

write(STDOUT\_FILENO, "Не удалось открыть семафоры", **sizeof**("Не удалось открыть семафоры"));

exit(EXIT\_FAILURE);

}

**char** input[BUFFER\_SIZE];

**while** (1) {

sem\_wait(sem\_child1);

strncpy(input, shared\_memory, BUFFER\_SIZE);

**if** (strlen(input) == 0) {

**break**; // Конец ввода

}

printf("Client1! Got string: <%s>\n", input);

**int** i = 0;

**while** (input[i] != '\0') {

**if** (strchr(vowels, input[i]) == NULL) {

write(file, &input[i], 1);

}

i++;

}

write(file, "\n", 1);

}

munmap(shared\_memory, BUFFER\_SIZE);

close(shm\_fd);

sem\_close(sem\_child1);

close(file);

}

**client2.c**

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#define BUFFER\_SIZE 1024

#define SHM\_NAME "/my\_shared\_memory"

#define SEM\_NAME\_CHILD2 "/my\_semaphore\_child2"

**int** main(**int** argc, **char** \*\*argv) {

**const** **char** \*vowels = "aeiouAEIOU";

printf("<<%s>>\n", argv[1]);

int32\_t file = open(argv[1], O\_WRONLY | O\_CREAT | O\_APPEND, 0600);

**if** (file == -1) {

**const** **char** msg[] = "error: failed to open requested file\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

// Открытие разделяемой памяти

**int** shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);

**if** (shm\_fd == -1) {

write(STDOUT\_FILENO, "Не удалось открыть разделяемую память", **sizeof**("Не удалось открыть разделяемую память"));

exit(EXIT\_FAILURE);

}

// Отображение разделяемой памяти

**char** \*shared\_memory = (**char** \*)mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

**if** (shared\_memory == MAP\_FAILED) {

write(STDOUT\_FILENO, "Не удалось отобразить разделяемую память", **sizeof**("Не удалось отобразить разделяемую память"));

exit(EXIT\_FAILURE);

}

// Открытие семафоров

sem\_t \*sem\_child2 = sem\_open(SEM\_NAME\_CHILD2, 0);

**if** (sem\_child2 == SEM\_FAILED) {

write(STDOUT\_FILENO, "Не удалось открыть семафоры", **sizeof**("Не удалось открыть семафоры"));

exit(EXIT\_FAILURE);

}

**char** input[BUFFER\_SIZE];

**while** (1) {

sem\_wait(sem\_child2);

strncpy(input, shared\_memory, BUFFER\_SIZE);

**if** (strlen(input) == 0) {

**break**; // Конец ввода

}

printf("Client2! Got string: <%s>\n", input);

**int** i = 0;

**while** (input[i] != '\0') {

**if** (strchr(vowels, input[i]) == NULL) {

write(file, &input[i], 1);

}

i++;

}

write(file, "\n", 1);

}

munmap(shared\_memory, BUFFER\_SIZE);

close(shm\_fd);

sem\_close(sem\_child2);

close(file);

}

**Протокол работы программы**

**Тестирование:**

$ ./t

file1.txt

file2.txt

621: I'm a child

622: I'm a child

<</mnt/c/Users/begemot/ClionProjects/OS-labs/Lab3/file2.txt>>

<</mnt/c/Users/begemot/ClionProjects/OS-labs/Lab3/file1.txt>>

string1

Client1! Got string: <string1>

someText

Client2! Got string: <someText>

andSomeMore

Client1! Got string: <andSomeMore>

Someone call 1234

Client2! Got string: <Someone call 1234>

mam

Client1! Got string: <mam>

papa

Client2! Got string: <papa>

**Strace:**

$ strace ./t

execve("./t", ["./t"], 0x7ffc8ac0b7a0 /\* 28 vars \*/) = 0

brk(NULL) = 0x55c3d8d32000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff0ec1f7000

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=20335, ...}) = 0

mmap(NULL, 20335, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7ff0ec1f2000

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) = 0x7ff0ebfe0000

mmap(0x7ff0ec008000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7ff0ec008000

mmap(0x7ff0ec190000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7ff0ec190000

mmap(0x7ff0ec1df000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7ff0ec1df000

mmap(0x7ff0ec1e5000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff0ec1e5000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff0ebfdd000

arch\_prctl(ARCH\_SET\_FS, 0x7ff0ebfdd740) = 0

set\_tid\_address(0x7ff0ebfdda10) = 630

set\_robust\_list(0x7ff0ebfdda20, 24) = 0

rseq(0x7ff0ebfde060, 0x20, 0, 0x53053053) = 0

mprotect(0x7ff0ec1df000, 16384, PROT\_READ) = 0

mprotect(0x55c3cb009000, 4096, PROT\_READ) = 0

mprotect(0x7ff0ec22f000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7ff0ec1f2000, 20335) = 0

readlink("/proc/self/exe", "/mnt/c/Users/begemot/ClionProjec"..., 1023) = 49

read(0, file1.txt

"file1.txt\n", 128) = 10

read(0, file2.txt

"file2.txt\n", 128) = 10

openat(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 3

ftruncate(3, 1024) = 0

mmap(NULL, 1024, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7ff0ec1f6000

openat(AT\_FDCWD, "/dev/shm/sem.my\_semaphore\_child1", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\xfd\x55\x26\x8e\xd5\x2a\xb8\x47", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.x9WsfR", 0x7ffe79db4a60, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.x9WsfR", O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 4

write(4, "\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, 4, 0) = 0x7ff0ec1f5000

link("/dev/shm/sem.x9WsfR", "/dev/shm/sem.my\_semaphore\_child1") = 0

fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

getrandom("\x73\x8a\xf5\x63\x85\xa5\xa1\x46", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55c3d8d32000

brk(0x55c3d8d53000) = 0x55c3d8d53000

unlink("/dev/shm/sem.x9WsfR") = 0

close(4) = 0

openat(AT\_FDCWD, "/dev/shm/sem.my\_semaphore\_child2", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\x9b\xc6\xd6\x04\x85\xea\xbf\x47", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.rpmZl5", 0x7ffe79db4a60, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.rpmZl5", O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 4

write(4, "\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, 4, 0) = 0x7ff0ec1f4000

link("/dev/shm/sem.rpmZl5", "/dev/shm/sem.my\_semaphore\_child2") = 0

fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

unlink("/dev/shm/sem.rpmZl5") = 0

close(4) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7ff0ebfdda10) = 631

631: I'm a child

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7ff0ebfdda10) = 632

632: I'm a child

read(0, <</mnt/c/Users/begemot/ClionProjects/OS-labs/Lab3/file1.txt>>

<</mnt/c/Users/begemot/ClionProjects/OS-labs/Lab3/file2.txt>>

string1

"string1\n", 1023) = 8

futex(0x7ff0ec1f5000, FUTEX\_WAKE, 1Client1! Got string: <string1>

) = 1

read(0, someText

"someText\n", 1023) = 9

futex(0x7ff0ec1f4000, FUTEX\_WAKE, 1) = 1

Client2! Got string: <someText>

read(0, andSomeMore

"andSomeMore\n", 1023) = 12

futex(0x7ff0ec1f5000, FUTEX\_WAKE, 1Client1! Got string: <andSomeMore>

) = 1

read(0, Someone call 1234

"Someone call 1234\n", 1023) = 18

futex(0x7ff0ec1f4000, FUTEX\_WAKE, 1Client2! Got string: <Someone call 1234>

) = 1

read(0, mam

"mam\n", 1023) = 4

futex(0x7ff0ec1f5000, FUTEX\_WAKE, 1Client1! Got string: <mam>

) = 1

read(0, papa

"papa\n", 1023) = 5

futex(0x7ff0ec1f4000, FUTEX\_WAKE, 1Client2! Got string: <papa>

) = 1

read(0, "", 1023) = 0

futex(0x7ff0ec1f5000, FUTEX\_WAKE, 1) = 1

futex(0x7ff0ec1f4000, FUTEX\_WAKE, 1) = 1

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 631

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=631, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=1 /\* 0.01 s \*/} ---

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=632, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7ff0ec1f6000, 1024) = 0

unlink("/dev/shm/my\_shared\_memory") = 0

munmap(0x7ff0ec1f5000, 32) = 0

munmap(0x7ff0ec1f4000, 32) = 0

unlink("/dev/shm/sem.my\_semaphore\_child1") = 0

unlink("/dev/shm/sem.my\_semaphore\_child2") = 0

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

**Программа демонстрирует взаимодействие между родительским и дочерними процессами с использованием разделяемой памяти и семафоров для синхронизации. Родительский процесс считывает строки от пользователя и передает их в разделяемую память, давая к ней доступ нужному дочернему процессу.**