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-215Б-23

Студент: Кобзев К. А.

Преподаватель: Миронов Е.С. (ПМИ)

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

Дата: 13.07.25

Москва, 2025

**Постановка задачи**

**Вариант 2.**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

2 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); — создаёт дочерний процесс.
* int execlp(const char\* file, const char\* arg, ...); — заменяет текущий образ процесса новым, ищет исполняемый файл в путях PATH.
* pid\_t wait(int\* stat\_loc); — ожидает завершения любого дочернего процесса.
* int shm\_open(const char \*name, int oflag, mode\_t mode); — создаёт или открывает объект разделяемой памяти POSIX.
* int ftruncate(int fd, off\_t length); — устанавливает размер файла (в данном случае, объекта разделяемой памяти).
* void\* mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset); — отображает файл или устройство в память, делая его доступным для процесса.
* int munmap(void \*addr, size\_t length); — отключает отображение области памяти.
* int shm\_unlink(const char \*name); — удаляет объект разделяемой памяти из системы.
* sem\_t\* sem\_open(const char \*name, int oflag, ...); — создаёт или открывает именованный семафор POSIX для синхронизации.
* int sem\_wait(sem\_t \*sem); — блокирует семафор (ожидает, пока его значение не станет > 0, а затем уменьшает его).
* int sem\_post(sem\_t \*sem); — освобождает семафор (увеличивает его значение), сигнализируя другому процессу.
* int sem\_close(sem\_t \*sem); — закрывает дескриптор именованного семафора.
* int sem\_unlink(const char \*name); — удаляет именованный семафор из системы.

Алгоритм решения

**Родитель (parent.c)**

1. Получает от пользователя имя файла для вывода.
2. Создаёт и настраивает объект разделяемой памяти с помощью shm\_open() и ftruncate().
3. Отображает этот объект в своё адресное пространство вызовом mmap().
4. Создаёт два именованных семафора с помощью sem\_open():

* sem\_write для контроля записи, с начальным значением 1 (запись разрешена).
* sem\_read для контроля чтения, с начальным значением 0 (читать нечего).

1. Создаёт дочерний процесс с помощью fork().
2. В родительском процессе, в цикле:

* Читает строку с числами от пользователя.
* Ожидает, пока дочерний процесс не освободит буфер, с помощью sem\_wait() на семафоре sem\_write.
* Копирует введённую строку в разделяемую память.
* Сигнализирует дочернему процессу о том, что данные готовы, с помощью sem\_post() на семафоре sem\_read.

1. При вводе пустой строки, записывает в разделяемую память пустую строку (как сигнал к завершению) и выходит из цикла.
2. Ожидает завершения дочернего процесса с помощью wait().
3. Освобождает все ресурсы: закрывает и удаляет семафоры (sem\_close, sem\_unlink), отключает (munmap) и удаляет (shm\_unlink) объект разделяемой памяти.

**Ребёнок (child.c)**

1. Открывает файл для записи, имя которого было получено как аргумент командной строки.
2. Открывает существующий объект разделяемой памяти (shm\_open()) и отображает его в своё адресное пространство (mmap()).
3. Открывает существующие именованные семафоры sem\_write и sem\_read по их именам.
4. В бесконечном цикле:

* Ожидает, пока родительский процесс не запишет данные, с помощью sem\_wait() на семафоре sem\_read.
* Читает данные из общего буфера в разделяемой памяти.
* Если строка пустая, выходит из цикла.
* Если в строке есть данные, вычисляет сумму всех чисел с плавающей точкой.
* Записывает полученную сумму в выходной файл.
* Сигнализирует родительскому процессу, что буфер был обработан и свободен для новой записи, с помощью sem\_post() на семафоре sem\_write.

1. После выхода из цикла закрывает семафоры и выходной файл, отключает разделяемую память (munmap) и завершает работу.

**Код программы**

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <semaphore.h>

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

#define SEM\_WRITE\_NAME "/my\_sem\_write"

#define SEM\_READ\_NAME "/my\_sem\_read"

#include "shared\_struct.h"

int main()

{

char filename[256];

printf("Введите имя файла для вывода: ");

if (!fgets(filename, sizeof(filename), stdin))

{

perror("fgets");

exit(EXIT\_FAILURE);

}

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

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

if (shm\_fd == -1)

{

perror("shm\_open");

exit(EXIT\_FAILURE);

}

if (ftruncate(shm\_fd, sizeof(struct shared\_data)) == -1)

{

perror("ftruncate");

exit(EXIT\_FAILURE);

}

struct shared\_data \*shared\_mem = mmap(NULL, sizeof(struct shared\_data),

PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shared\_mem == MAP\_FAILED)

{

perror("mmap");

exit(EXIT\_FAILURE);

}

close(shm\_fd);

// Удаляем старые семафоры, если они остались от предыдущего запуска

sem\_unlink(SEM\_WRITE\_NAME);

sem\_unlink(SEM\_READ\_NAME);

// Создаем семафор для записи

sem\_t \*sem\_write = sem\_open(SEM\_WRITE\_NAME, O\_CREAT, 0666, 1);

if (sem\_write == SEM\_FAILED)

{

perror("sem\_open write");

exit(EXIT\_FAILURE);

}

// Создаем семафор для чтения

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

if (sem\_read == SEM\_FAILED)

{

perror("sem\_open read");

exit(EXIT\_FAILURE);

}

pid\_t pid = fork();

if (pid < 0)

{

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0)

{

execlp("./child", "child", filename, NULL);

perror("execlp");

exit(EXIT\_FAILURE);

}

else

{

printf("Введите строки с числами (float). Пустая строка — завершение.\n");

while (1)

{

char buffer[BUFFER\_SIZE];

if (!fgets(buffer, sizeof(buffer), stdin) || buffer[0] == '\n')

{

sem\_wait(sem\_write);

shared\_mem->buffer[0] = '\0';

sem\_post(sem\_read);

break;

}

sem\_wait(sem\_write);

strncpy(shared\_mem->buffer, buffer, BUFFER\_SIZE);

sem\_post(sem\_read);

}

wait(NULL);

}

sem\_close(sem\_write);

sem\_close(sem\_read);

sem\_unlink(SEM\_WRITE\_NAME);

sem\_unlink(SEM\_READ\_NAME);

munmap(shared\_mem, sizeof(struct shared\_data));

shm\_unlink(SHM\_NAME);

return 0;

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <semaphore.h>

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

#define SEM\_WRITE\_NAME "/my\_sem\_write"

#define SEM\_READ\_NAME "/my\_sem\_read"

#include "shared\_struct.h"

int main(int argc, char \*argv[])

{

if (argc != 2)

{

fprintf(stderr, "Использование: %s <имя\_файла\_для\_вывода>\n", argv[0]);

return 1;

}

FILE \*outFile = fopen(argv[1], "w");

if (!outFile)

{

perror("Ошибка открытия файла для записи");

return 1;

}

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

if (shm\_fd == -1)

{

perror("shm\_open child");

fclose(outFile);

exit(EXIT\_FAILURE);

}

struct shared\_data \*shared\_mem = mmap(NULL, sizeof(struct shared\_data),

PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shared\_mem == MAP\_FAILED)

{

perror("mmap child");

fclose(outFile);

close(shm\_fd);

exit(EXIT\_FAILURE);

}

close(shm\_fd);

sem\_t \*sem\_write = sem\_open(SEM\_WRITE\_NAME, 0);

if (sem\_write == SEM\_FAILED)

{

perror("sem\_open write child");

exit(EXIT\_FAILURE);

}

sem\_t \*sem\_read = sem\_open(SEM\_READ\_NAME, 0);

if (sem\_read == SEM\_FAILED)

{

perror("sem\_open read child");

exit(EXIT\_FAILURE);

}

while (1)

{

sem\_wait(sem\_read);

if (shared\_mem->buffer[0] == '\0')

{

sem\_post(sem\_write);

break;

}

double sum = 0.0f;

char \*ptr = shared\_mem->buffer;

char \*endptr;

while (\*ptr)

{

float num = strtof(ptr, &endptr);

if (ptr == endptr)

{

if (\*ptr == '\0' || \*ptr == '\n')

break;

ptr++;

}

else

{

sum += num;

ptr = endptr;

}

}

fprintf(outFile, "%f\n", sum);

fflush(outFile);

sem\_post(sem\_write);

}

sem\_close(sem\_write);

sem\_close(sem\_read);

fclose(outFile);

munmap(shared\_mem, sizeof(struct shared\_data));

return 0;

}

**shared\_struct.h**

#define BUFFER\_SIZE 1024

struct shared\_data {

char buffer[BUFFER\_SIZE];

};

**Протокол работы программы**

**Тестирование:**

➜ src ./parent

Введите имя файла для вывода: result.txt

Введите строки с числами (float). Пустая строка — завершение.

1.2 3.0

0.0 1.2 3.0

➜ src cat result.txt

4.200000

4.200000

➜ src ./parent

Введите имя файла для вывода: result.txt

Введите строки с числами (float). Пустая строка — завершение.

1 2.3 100

0 2.323 1212

➜ src cat result.txt

103.300000

1214.323000

**strace**

execve("./parent", ["./parent"], 0xffffd75a0228 /\* 12 vars \*/) = 0

brk(NULL) = 0x35959000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xffffa36f0000

faccessat(AT\_FDCWD, "/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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=25959, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 25959, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0xffffa36e9000

close(3) = 0

openat(AT\_FDCWD, "/lib/aarch64-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\267\0\1\0\0\0000y\2\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1651408, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 1826912, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xffffa34f8000

mmap(0xffffa3500000, 1761376, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0) = 0xffffa3500000

munmap(0xffffa34f8000, 32768) = 0

munmap(0xffffa36af000, 28768) = 0

mprotect(0xffffa3687000, 86016, PROT\_NONE) = 0

mmap(0xffffa369c000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18c000) = 0xffffa369c000

mmap(0xffffa36a2000, 49248, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0xffffa36a2000

close(3) = 0

set\_tid\_address(0xffffa36f1050) = 31

set\_robust\_list(0xffffa36f1060, 24) = 0

rseq(0xffffa36f16a0, 0x20, 0, 0xd428bc00) = 0

mprotect(0xffffa369c000, 16384, PROT\_READ) = 0

mprotect(0x41f000, 4096, PROT\_READ) = 0

mprotect(0xffffa36f5000, 8192, PROT\_READ) = 0

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

munmap(0xffffa36e9000, 25959) = 0

newfstatat(1, "", {st\_mode=S\_IFREG|0644, st\_size=1819, ...}, AT\_EMPTY\_PATH) = 0

getrandom("\xac\x55\x86\x13\x2c\x2f\x48\x5a", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x35959000

brk(0x3597a000) = 0x3597a000

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

read(0, "result.txt\n", 1024) = 11

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) = 0xffffa36ef000

close(3) = 0

unlinkat(AT\_FDCWD, "/dev/shm/sem.my\_sem\_write", 0) = -1 ENOENT (No such file or directory)

unlinkat(AT\_FDCWD, "/dev/shm/sem.my\_sem\_read", 0) = -1 ENOENT (No such file or directory)

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

getrandom("\x54\x94\xab\x3e\x1a\x5f\x51\x80", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.q6oG8C", 0xffffdf02c638, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.q6oG8C", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 3

write(3, "\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, 3, 0) = 0xffffa36ee000

linkat(AT\_FDCWD, "/dev/shm/sem.q6oG8C", AT\_FDCWD, "/dev/shm/sem.my\_sem\_write", 0) = 0

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

unlinkat(AT\_FDCWD, "/dev/shm/sem.q6oG8C", 0) = 0

close(3) = 0

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

getrandom("\x60\x79\xd4\x93\x9f\xa9\xc1\x98", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.ofnd2n", 0xffffdf02c638, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.ofnd2n", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 3

write(3, "\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, 3, 0) = 0xffffa36ed000

linkat(AT\_FDCWD, "/dev/shm/sem.ofnd2n", AT\_FDCWD, "/dev/shm/sem.my\_sem\_read", 0) = 0

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

unlinkat(AT\_FDCWD, "/dev/shm/sem.ofnd2n", 0) = 0

close(3) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0xffffa36f1050) = 32

strace: Process 32 attached

[pid 31] read(0, <unfinished ...>

[pid 32] set\_robust\_list(0xffffa36f1060, 24) = 0

[pid 32] execve("./child", ["child", "result.txt"], 0xffffdf02d068 /\* 12 vars \*/) = 0

[pid 32] brk(NULL) = 0x294dd000

[pid 32] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xffff9933d000

[pid 32] faccessat(AT\_FDCWD, "/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

[pid 32] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

[pid 32] newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=25959, ...}, AT\_EMPTY\_PATH) = 0

[pid 32] mmap(NULL, 25959, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0xffff99336000

[pid 32] close(3) = 0

[pid 32] openat(AT\_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

[pid 32] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0000y\2\0\0\0\0\0"..., 832) = 832

[pid 32] newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1651408, ...}, AT\_EMPTY\_PATH) = 0

[pid 32] mmap(NULL, 1826912, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xffff99145000

[pid 32] mmap(0xffff99150000, 1761376, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0) = 0xffff99150000

[pid 32] munmap(0xffff99145000, 45056) = 0

[pid 32] munmap(0xffff992ff000, 16480) = 0

[pid 32] mprotect(0xffff992d7000, 86016, PROT\_NONE) = 0

[pid 32] mmap(0xffff992ec000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18c000) = 0xffff992ec000

[pid 32] mmap(0xffff992f2000, 49248, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0xffff992f2000

[pid 32] close(3) = 0

[pid 32] set\_tid\_address(0xffff9933e050) = 32

[pid 32] set\_robust\_list(0xffff9933e060, 24) = 0

[pid 32] rseq(0xffff9933e6a0, 0x20, 0, 0xd428bc00) = 0

[pid 32] mprotect(0xffff992ec000, 16384, PROT\_READ) = 0

[pid 32] mprotect(0x41f000, 4096, PROT\_READ) = 0

[pid 32] mprotect(0xffff99342000, 8192, PROT\_READ) = 0

[pid 32] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 32] munmap(0xffff99336000, 25959) = 0

[pid 32] getrandom("\xf7\x50\xe0\x1a\x1f\x50\x15\x5e", 8, GRND\_NONBLOCK) = 8

[pid 32] brk(NULL) = 0x294dd000

[pid 32] brk(0x294fe000) = 0x294fe000

[pid 32] openat(AT\_FDCWD, "result.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0666) = 3

[pid 32] openat(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 4

[pid 32] mmap(NULL, 1024, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0xffff9933c000

[pid 32] close(4) = 0

[pid 32] openat(AT\_FDCWD, "/dev/shm/sem.my\_sem\_write", O\_RDWR|O\_NOFOLLOW) = 4

[pid 32] newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

[pid 32] mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0xffff9933b000

[pid 32] close(4) = 0

[pid 32] openat(AT\_FDCWD, "/dev/shm/sem.my\_sem\_read", O\_RDWR|O\_NOFOLLOW) = 4

[pid 32] newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

[pid 32] mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0xffff9933a000

[pid 32] close(4) = 0

[pid 32] futex(0xffff9933a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 31] <... read resumed>"1.0 2.0\n", 1024) = 8

[pid 31] futex(0xffffa36ed000, FUTEX\_WAKE, 1) = 1

[pid 32] <... futex resumed>) = 0

[pid 31] read(0, <unfinished ...>

[pid 32] newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=0, ...}, AT\_EMPTY\_PATH) = 0

[pid 32] write(3, "3.000000\n", 9) = 9

[pid 32] futex(0xffff9933a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 31] <... read resumed>"0.0\n", 1024) = 4

[pid 31] futex(0xffffa36ed000, FUTEX\_WAKE, 1) = 1

[pid 31] read(0, <unfinished ...>

[pid 32] <... futex resumed>) = 0

[pid 32] write(3, "0.000000\n", 9) = 9

[pid 32] futex(0xffff9933a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 31] <... read resumed>"\n", 1024) = 1

[pid 31] futex(0xffffa36ed000, FUTEX\_WAKE, 1) = 1

[pid 31] wait4(-1, <unfinished ...>

[pid 32] <... futex resumed>) = 0

[pid 32] munmap(0xffff9933b000, 32) = 0

[pid 32] munmap(0xffff9933a000, 32) = 0

[pid 32] close(3) = 0

[pid 32] munmap(0xffff9933c000, 1024) = 0

[pid 32] exit\_group(0) = ?

[pid 32] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 32

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=32, si\_uid=0, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0xffffa36ee000, 32) = 0

munmap(0xffffa36ed000, 32) = 0

unlinkat(AT\_FDCWD, "/dev/shm/sem.my\_sem\_write", 0) = 0

unlinkat(AT\_FDCWD, "/dev/shm/sem.my\_sem\_read", 0) = 0

munmap(0xffffa36ef000, 1024) = 0

unlinkat(AT\_FDCWD, "/dev/shm/my\_shared\_memory", 0) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 161Введите имя файла для вывода: Введите строки с числами (float). Пустая строка — завершение.

) = 161

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В результате выполнения лабораторной работы была разработана и отлажена программа, демонстрирующая взаимодействие процессов посредством технологии отображаемых в память файлов (memory-mapped files). Для создания дочернего процесса использовался системный вызов fork(), а для организации области разделяемой памяти — функции shm\_open(), ftruncate() и mmap().