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-209БВ-24

Студент: Кобзев К. А.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.11.25

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).
5. Создаёт дочерний процесс с помощью fork().
6. 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.

7. При вводе пустой строки, записывает в разделяемую память пустую строку (как сигнал к завершению) и выходит из цикла.
8. Ожидает завершения дочернего процесса с помощью wait().
9. 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.
5. 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\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, 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\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, 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\3\0\267\0\1\0\0\0000y\2\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().