

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: Тропн М.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 02.10.24

Москва, 2024

# Постановка задачи

## Вариант 7.

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством shared memory и memory mapping

Задание

7 вариант) В файле записаны команды вида: «число число число<newline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

## Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void);` – создает дочерний процесс.
- `int shm_open(const char *__name, int __oflag, mode_t __mode)` — открывает сегмент shm
- `void *mmap(void *__addr, size_t __len, int __prot, int __flags, int __fd, off_t __offset)` — создает новый маппинг в виртуальном адресном пространстве
- `sem_t *sem_open (const char *__name, int __oflag, ...)` - открывает именованный семафор
- `int sem_unlink (const char *__name)` — удаляет именованный семафор
- `int sem_wait(sem_t *sem)` - уменьшает (блокирует) семафор
- `int sem_post(sem_t *sem)` - увеличивает (разблокирует) семафор
- `int open(const char *pathname, int flags, mode_t mode)` - открытие\создание файла
- `int close(int fd)` - закрыть файл
- `void exit(int status)` - завершения выполнения процесса и возвращение статуса
- `int execv(const char *filename, char *const argv[])` - замена образа памяти процесса
- `pid_t getpid(void)` — получение ID процесса
- `ssize_t read(int __fd, void* __buf, size_t __nbytes)` — чтение из fd в буфер
- `ssize_t write(int __fd, const void* __buf, size_t __n)` — запись байтов в буфер

Главная программа создает дочерний процесс. Он открывает файл на чтение. Создаётся несколько shm: для записи результата, передачи входного файла и размера. Создаётся mmap для считывания и записи результата и входного потока. Размер передается через `int*`, поэтому в mmap не нуждается, строки же нуждаются. Для синхронизации записи и чтения используется семафор.

## Код программы

server.c

```
/**
 * @file server.c
 * @author your name (you@domain.com)
 * @brief
 * @version 0.1
 * @date 2024-10-02
 *
 * @copyright Copyright (c) 2024
 *
 */

#include "pool.h"

#include <fcntl.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

static char CLIENT_PROGRAM_NAME[] = "client";

int main(int argc, char** argv) {
    if (argc != 2) {
        _print(ERROR, "usage: %s filename\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // хде я?
```

```

char prospath[1024] = {0};

{

    // собственно узнаем где

    ssize_t len = readlink("/proc/self/exe", prospath, sizeof(prospath) - 1);

    if (len == -1) {

        _print(ERROR, "error: failed to read full program path\n");

        exit(EXIT_FAILURE);

    }


    // всё лишнее убираем

    while (prospath[len] != '/')

        --len;


    prospath[len] = '\0';

}


// создаем новый процесс

const pid_t child = fork();


switch (child) {

    case -1: { // обработка ошибки

        _print(ERROR, "error: failed to spawn new process\n");

        exit(EXIT_FAILURE);

    } break;


    case 0: { // Я - ребенок и не знаю свой PID

        // собственно узнаю


        /* файл*/

        int file = open(argv[1], O_RDONLY);

        if (file == -1) {

            perror("Ошибка при открытии файла");

            exit(EXIT_FAILURE);

        }

    }

}

```

```
}
```

```
struct stat file_stat;
```

```
if (fstat(file, &file_stat) < 0) {
```

```
    const char msg[] = "error: failed create stat for file\n";
```

```
    write(STDERR_FILENO, msg, sizeof(msg));
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
```

```
if (shm_fd == -1) {
```

```
    _print(ERROR, "%s: open failed\n", SHM_NAME);
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
// укорачиваем файла строго до определенной длины
```

```
if (ftruncate(shm_fd, file_stat.st_size) == -1) {
```

```
    _print(ERROR, "%s: ftruncate failed\n", SHM_NAME);
```

```
    close(shm_fd);
```

```
    shm_unlink(SHM_NAME);
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
char* src = mmap(NULL, file_stat.st_size, PROT_READ | PROT_WRITE,
```

```
    MAP_SHARED, shm_fd, 0);
```

```
if (src == MAP_FAILED) {
```

```
    _print(ERROR, "%s: mapping failed\n", SHM_NAME);
```

```
    close(shm_fd);
```

```
    shm_unlink(SHM_NAME);
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
// char buf[BUFSIZ];
```

```
read(file, src, file_stat.st_size);
```

```

// strcpy(buf, src);

// fprintf(stdout, "%s\n", buf);

int* FILE_SIZE = create_mmap_int("/file_size_shm\0");


*FILE_SIZE = file_stat.st_size;

close(file);


/* ГОВОРИМ О ГОТОВНОСТИ ВХОДНОГО ПОТОКА */

sem_t* sem_parent_ready = sem_open(SEM_PARENT_READY, O_CREAT, 0666, 0);

if (sem_parent_ready == SEM_FAILED) {

    perror("sem_open parent");

    exit(EXIT_FAILURE);

}

sem_post(sem_parent_ready);


{

    char path[1024];

    snprintf(path, sizeof(path) - 1, "%s/%s", proppath,

             CLIENT_PROGRAM_NAME);


    // аргументами для клиента являются -- сам клиент (его запускатор)б

    // полезные аргументы (файл)б и терминатор, ведь exes требует список с

    // терминирующем нулем

    char* const args[] = {CLIENT_PROGRAM_NAME, argv[1], NULL};


    int32_t status = execv(path, args);


    if (status == -1) {

        const char msg[] =

            "error: failed to exec into new executable image\n";

        write(STDERR_FILENO, msg, sizeof(msg));

        exit(EXIT_FAILURE);

    }
}

```

```

    }

} break;

default: { // Я родитель и знаю PID дочерний

    // pid_t pid = getpid(); // Получаем родительский PID

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

    // блокируем родителя до конца выполнения дочерних процессов

    int child_status;

    wait(&child_status);

    sem_t* sem_child_ready = sem_open(SEM_CHILD_READY, 0);

    sem_wait(sem_child_ready);

    // чтение ответа

    int shm_fd;

    if ((shm_fd = shm_open(RES_SHM, O_RDONLY, 0666)) == -1) {

        _print(ERROR, "error: shm_open parent\n");

        // close(shm_fd);

        // shm_unlink(SHM_NAME);

        // shm_unlink(RES_SHM);

        sem_unlink(SEM_PARENT_READY);

        sem_unlink(SEM_CHILD_READY);

        sem_close(sem_child_ready);

        exit(EXIT_FAILURE);

    }

    ftruncate(shm_fd, BUFSIZ);

    char* res;

    if ((res = mmap(0, BUFSIZ, PROT_READ, MAP_SHARED, shm_fd, 0)) ==

        MAP_FAILED) {

        _print(ERROR, "error: res mmap parent\n");

        close(shm_fd);

```

```

shm_unlink(SHM_NAME);

shm_unlink(RES_SHM);

sem_unlink(SEM_PARENT_READY);

sem_unlink(SEM_CHILD_READY);
sem_close(sem_child_ready);
exit(EXIT_FAILURE);
}

_print(SUCCESS, "res=%s\n", res);

if (child_status != EXIT_SUCCESS) {
    const char msg[] = "error: child exited with error\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    perror("");
    close(shm_fd);
    shm_unlink(SHM_NAME);
    shm_unlink(RES_SHM);
    sem_unlink(SEM_PARENT_READY);
    sem_unlink(SEM_CHILD_READY);
    sem_close(sem_child_ready);
    exit(child_status);
}

close(shm_fd);

shm_unlink(SHM_NAME);
shm_unlink(RES_SHM);
sem_unlink(SEM_PARENT_READY);
sem_unlink(SEM_CHILD_READY);
sem_close(sem_child_ready);
} break;
}
}

```

**client.c**



```

/**
 * @file client.c
 * @author your name (you@domain.com)
 * @brief
 * @version 0.1
 * @date 2024-10-02
 *
 * @copyright Copyright (c) 2024
 *
 */
#include <ctype.h>
#include <math.h>
#include <stdbool.h>
#include <stdint.h>

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

#include "pool.h"

// extern int FILE_SIZE;

float summ(const char* src) {
    if (src == NULL) {
        return 0.0;
    }
    float sum = 0.0;
    char* endptr;

    while (*src) {
        while (*src && !isdigit(*src) && *src != '.' && *src != '-' &&
            *src != '+') {
            ++src;
        }

        float value = strtod(src, &endptr);
        if (value == HUGE_VAL || value == -HUGE_VAL) {
            return 0.0;
        }

        if (endptr != src) {
            sum += value;
        }
    }
}

```

```

        src = endptr;
    } else {
        ++src;
    }
}

return sum;
}

int main(int argc, char** argv) {
    /* Получение сигнала о том, что записано */
    sem_t* sem_parent_ready = sem_open(SEM_PARENT_READY, 0);
    sem_wait(sem_parent_ready);
    /* возврат размера файла */
    int tmp = shm_open("/file_size_shm\0", O_RDWR, 0666);
    if (tmp == -1) {
        _print(ERROR, "%s: open failed child\n", "/file_size_shm\0");
        exit(EXIT_FAILURE);
    }
    int* shared_variable = mmap(0, sizeof(int), PROT_READ, MAP_SHARED, tmp, 0);
    if (shared_variable == MAP_FAILED) {
        _print(ERROR, "%s: mapping failed child\n", "/file_size_shm\0");
        exit(EXIT_FAILURE);
    }

    int FILE_SIZE = *shared_variable;

    destroy_mmap_int(shared_variable, "/file_size_shm\0");
    /*=====*/
    /* работа с файлом (входной поток) */
    int shm_fd;
    if ((shm_fd = shm_open(SHM_NAME, O_RDONLY, 0666)) == -1) {
        _print(ERROR, "error: shm_open child\n");
        exit(EXIT_FAILURE);
    }

    char* src = mmap(0, FILE_SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
    if (src == MAP_FAILED) {
        _print(ERROR, "%s: mapping failed child\n", SHM_NAME);
        exit(EXIT_FAILURE);
    }

    float sum = summ(src);
    /*=====*/
    /* запись в результирующую память */
    int res_fd;
    if ((res_fd = shm_open(RES_SHM, O_CREAT | O_RDWR, 0666)) == -1) {

```

```

    _print(ERROR, "error: shm_open child\n");
    exit(EXIT_FAILURE);
}

```

```

ftruncate(res_fd, BUFSIZ);

```

```

void* ressrc = mmap(0, BUFSIZ, PROT_READ | PROT_WRITE, MAP_SHARED, res_fd, 0);
if (ressrc == MAP_FAILED) {
    perror("mmap");
    exit(EXIT_FAILURE);
}

```

```

if (0 == sprintf(ressrc, "%f", sum)) {
    perror("write result");
    exit(EXIT_FAILURE);
}

```

```

/* уведомление о готовности результата */

```

```

sem_t* sem_child_ready = sem_open(SEM_CHILD_READY, O_CREAT, 0666, 0);
sem_post(sem_child_ready);

```

```

/*=====*/

```

```

if (-1 == munmap(src, FILE_SIZE)) {
    perror("munmap");
    exit(EXIT_FAILURE);
}

```

```

if (-1 == munmap(ressrc, BUFSIZ)) {
    perror("munmap");
    exit(EXIT_FAILURE);
}

```

```

close(shm_fd);
close(res_fd);
sem_close(sem_parent_ready);
exit(EXIT_SUCCESS);
}

```

## **pool.h**

```

extern int FILE_SIZE;

```

```

#pragma once

```

```

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

```

```

#define SEM_PARENT_READY "/sem_parent_ready"
#define SEM_CHILD_READY "/sem_child_ready"
#define SHM_NAME "/filesshm\0"
#define RES_SHM "/resshm\0"

// размер входного файла и shm, должен быть протянут и в client.c

typedef enum PRINT_MODES {
    SUCCESS,
    ERROR,
} PRINT_MODES;

int _print(char mode, char* fmt, ...) {
    if (fmt == NULL) {
        write(STDERR_FILENO, "ERROR", 6);
    }
    va_list vars;
    va_start(vars, fmt);
    char msg[BUFSIZ];
    vsprintf(msg, fmt, vars);
    write(mode == ERROR ? STDERR_FILENO : STDOUT_FILENO, msg, strlen(msg));
    va_end(vars);
    return 0;
}

int* create_mmap_int(const char* name) {
    int new_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    ftruncate(new_fd, sizeof(int));

    int* FILE_SIZE =
        mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, new_fd, 0);
    return FILE_SIZE;
}

void destroy_mmap_int(int* var, const char* name) {
    munmap(var, sizeof(int));
    shm_unlink(name);
}

```

## Протокол работы программы

### Тестирование:

```

lemito@lemito:~/Desktop/OSi/lab1$ cat data.base
52.25 48.50 7878.0
lemito@lemito:~/Desktop/OSi/lab1$ ./server data.base
7978.750000

```

```
lemito@lemito:~/Desktop/OSi/lab1$ cat data.base
0.0 a b c
lemito@lemito:~/Desktop/OSi/lab1$ ./server data.base
0.000000
```

### Strace:

```
execve("/home/lemito/Desktop/OSi/lab3/out/build/111/server", ["/home/lemito/Desktop/OSi/lab3/ou"... , "data.base"],
0x7fff0e56cdb0 /* 119 vars */) = 0

brk(NULL)                = 0x568f40af7000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7499367fe000

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=98319, ...}) = 0

mmap(NULL, 98319, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7499367e5000

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\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) = 0x749936400000

mmap(0x749936428000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x749936428000

mmap(0x7499365b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000)
= 0x7499365b0000

mmap(0x7499365ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7499365ff000

mmap(0x749936605000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x749936605000

close(3)                   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7499367e2000
```

```

arch_prctl(ARCH_SET_FS, 0x7499367e2740) = 0

set_tid_address(0x7499367e2a10)      = 81598

set_robust_list(0x7499367e2a20, 24)   = 0

rseq(0x7499367e3060, 0x20, 0, 0x53053053) = 0

mprotect(0x7499365ff000, 16384, PROT_READ) = 0

mprotect(0x568f3ec5b000, 4096, PROT_READ) = 0

mprotect(0x749936836000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7499367e5000, 98319)         = 0

readlink("/proc/self/exe", "/home/lemite/Desktop/OSi/lab3/ou"..., 1023) = 50

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7499367e2a10) = 81599

strace: Process 81599 attached

[pid 81598] wait4(-1, <unfinished ...>

[pid 81599] set_robust_list(0x7499367e2a20, 24) = 0

[pid 81599] openat(AT_FDCWD, "data.base", O_RDONLY) = 3

[pid 81599] fstat(3, {st_mode=S_IFREG|0664, st_size=5, ...}) = 0



[pid 81599] openat(AT_FDCWD, "/dev/shm/filesshm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4



[pid 81599] ftruncate(4, 5)          = 0



[pid 81599] mmap(NULL, 5, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7499367fd000



[pid 81599] read(3, "52 25", 5)      = 5



[pid 81599] openat(AT_FDCWD, "/dev/shm/file_size_shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 5



[pid 81599] ftruncate(5, 4)          = 0

[pid 81599] mmap(NULL, 4, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7499367fc000

[pid 81599] close(3)                 = 0

[pid 81599] openat(AT_FDCWD, "/dev/shm/sem.sem_parent_ready", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

[pid 81599] getrandom("\x79\xda\xa7\x8f\xf4\xe1\x10\x90", 8, GRND_NONBLOCK) = 8

```

```
[pid 81599] newfstatat(AT_FDCWD, "/dev/shm/sem.t9QtFp", 0x7ffc033aeae0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
```

```
[pid 81599] openat(AT_FDCWD, "/dev/shm/sem.t9QtFp", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|
O_CLOEXEC, 0666) = 3
```

[illegible]

```
[pid 81599] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7499367fb000
```

```
[pid 81599] link("/dev/shm/sem.t9QtFp", "/dev/shm/sem.sem_parent_ready") = 0
```

```
[pid 81599] fstat(3, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0
```

```
[pid 81599] getrandom("\x91\xbb\x17\xf6\xb5\x4c\x37\xc0", 8, GRND_NONBLOCK) = 8
```

```
[pid 81599] brk(NULL)                = 0x568f40af7000
```

```
[pid 81599] brk(0x568f40b18000) = 0x568f40b18000
```

```
[pid 81599] unlink("/dev/shm/sem.t9QtFp") = 0
```

```
[pid 81599] close(3)          = 0
```

```
[pid 81599] execve("/home/lemto/Desktop/OSi/lab3/out/build/111/client", ["client", "data.base"], 0x7ffc033af900 /*
119 vars */) = 0
```

```
[pid 81599] brk(NULL)           = 0x63d023523000
```

```
[pid 81599] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x71a6f1c2a000
```

```
[pid 81599] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
```

```
[pid 81599] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

```
[pid 81599] fstat(3, {st_mode=S_IFREG|0644, st_size=98319, ...}) = 0
```

```
[pid 81599] mmap(NULL, 98319, PROT_READ, MAP_PRIVATE, 3, 0) = 0x71a6f1c11000
```

```
[pid 81599] close(3) = 0
```

```
[pid 81599] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

```
[pid 81599] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
```

```
[pid 81599] pread64(3, "\x06\x00\x00\x04\x00\x00@\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00"..., 784, 64) = 784
```

```
[pid 81599] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
```

```
[pid 81599] 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
```

```
[pid 81599] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x71a6f1800000
```

[pid 81599] mmap(0x71a6f1828000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x71a6f1828000

[pid 81599] mmap(0x71a6f19b0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x71a6f19b0000

[pid 81599] mmap(0x71a6f19ff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x71a6f19ff000

[pid 81599] mmap(0x71a6f1a05000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x71a6f1a05000

[pid 81599] close(3) = 0

[pid 81599] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x71a6f1c0e000

[pid 81599] arch\_prctl(ARCH\_SET\_FS, 0x71a6f1c0e740) = 0

[pid 81599] set\_tid\_address(0x71a6f1c0ea10) = 81599

[pid 81599] set\_robust\_list(0x71a6f1c0ea20, 24) = 0

[pid 81599] rseq(0x71a6f1c0f060, 0x20, 0, 0x53053053) = 0

[pid 81599] mprotect(0x71a6f19ff000, 16384, PROT\_READ) = 0

[pid 81599] mprotect(0x63d022bf4000, 4096, PROT\_READ) = 0

[pid 81599] mprotect(0x71a6f1c62000, 8192, PROT\_READ) = 0

[pid 81599] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 81599] munmap(0x71a6f1c11000, 98319) = 0

**[pid 81599] openat(AT\_FDCWD, "/dev/shm/sem.sem\_parent\_ready", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 3**

[pid 81599] fstat(3, {st\_mode=S\_IFREG|0664, st\_size=32, ...}) = 0

[pid 81599] getrandom("\x9f\x12\x4c\x92\x57\xce\x78\x64", 8, GRND\_NONBLOCK) = 8

[pid 81599] brk(NULL) = 0x63d023523000

[pid 81599] brk(0x63d023544000) = 0x63d023544000

**[pid 81599] mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x71a6f1c29000**

[pid 81599] close(3) = 0

**[pid 81599] openat(AT\_FDCWD, "/dev/shm/file\_size\_shm", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 3**

[pid 81599] mmap(NULL, 4, PROT\_READ, MAP\_SHARED, 3, 0) = 0x71a6f1c28000



```
<... wait4 resumed>[ {WIFEXITED(s) && WEXITSTATUS(s) == 0}, 0, NULL) = 81599
```

```

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=81599, si_uid=1000, si_status=0, si_etime=0,
si_stime=0} ---

openat(AT_FDCWD, "/dev/shm/sem.sem_child_ready", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0

getrandom("\xdc\xdc\xdc\xf8\xe\xdd\xf7\x69", 8, GRND_NONBLOCK) = 8

brk(NULL)                                = 0x568f40af7000

brk(0x568f40b18000)                      = 0x568f40b18000

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7499367fd000

close(3)                                = 0

openat(AT_FDCWD, "/dev/shm/resshm", O_RDONLY|O_NOFOLLOW|O_CLOEXEC) = 3

ftruncate(3, 8192)                       = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ, MAP_SHARED, 3, 0) = 0x7499367fb000

write(1, "res=77.000000\n", 14res=77.000000

)    = 14

close(3)                                = 0

unlink("/dev/shm/filesshm")              = 0

unlink("/dev/shm/resshm")                = 0

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

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

munmap(0x7499367fd000, 32)              = 0

exit_group(0)                            = ?

+++ exited with 0 +++

```

## Вывод

В ходе лабораторной работы я приобрел практические навыки в управлении процессами ОС и обеспечении обмена данных между процессами посредством shared memory и mmap. Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. Проблем в ходе выполнения не возникло.