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-24

Студент: Ельцова Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 30.10.25

Москва, 2024

Постановка задачи

Вариант 10.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

В данной лабораторной работе использован POSIX Threads для создания многопоточной программы.

Основные использованные системные вызовы и функции:

- pid_t fork(void); – создаёт копию процесса (дочерний процесс).
- int execv(const char *path, char *const argv[]); – заменяет образ текущей программы на указанную, принимая аргументы в виде массива.
- pid_t waitpid(pid_t pid, int *status, int options); – ожидает завершения дочернего процесса.
- pid_t getpid(void); – возвращает идентификатор текущего процесса (PID).
- int shm_open(const char *name, int oflag, mode_t mode); – создаёт/открывает объект разделяемой памяти.
- int ftruncate(int fd, off_t length); – устанавливает размер shared memory.
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset); – отображает shared memory в адресное пространство процесса.
- int munmap(void *addr, size_t length); – удаляет отображение shared memory.
- int shm_unlink(const char *name); – удаляет объект shared memory из системы.
- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – создаёт/открывает именованный семафор.
- int sem_wait(sem_t *sem); – уменьшает значение семафора (блокирующая операция).
- int sem_post(sem_t *sem); – увеличивает значение семафора.
- int sem_close(sem_t *sem); – закрывает семафор.
- int sem_unlink(const char *name); – удаляет именованный семафор из системы.

Описание программы

Входные данные: Программа принимает 1 аргумент командной строки – имя файла с числами для обработки.

Основные шаги работы программы:

1. Создание ресурсов межпроцессного взаимодействия. Сервер создает именованные объекты shared memory и семафоры с уникальными именами на основе своего PID.

Используются два семафора для синхронизации: sem_req - управляет доступом к буферу (начальное значение = 1); sem_res - управляет готовностью данных (начальное значение = 0).

2. Запуск клиентского процесса. Сервер создает дочерний процесс с помощью fork(). Дочерний процесс заменяет свой образ на клиентскую программу через execv(). Сервер передает клиенту имя файла и свой PID для подключения к созданным ресурсам.
3. Обработка чисел.
   Алгоритм работы клиента:
   - Открывает файл с числами и читает его построчно
   - Для каждого числа выполняет цикл:
     - Захватывает буфер через sem_wait(sem_req)
     - Записывает число в shared memory
     - Сигнализирует серверу через sem_post(sem_res)
     - Ждет ответ через sem_wait(sem_req)
     - Если получено корректное число - выводит его
     - Освобождает буфер через sem_post(sem_req)

   Алгоритм работы сервера:
   - Входит в основной цикл обработки.
   - Ожидает данные от клиента через sem_wait(sem_res)
   - Для каждого полученного числа:
     - Парсит число из строки
     - Проверяет условия завершения (отрицательное или простое число)
     - Если число не простое и положительное – формирует ответ
     - Освобождает буфер через sem_post(sem_req)
4. Завершение работы. Условия завершения: отрицательное число, простое число, конец файла, сигнал UINT32_MAX в shared memory. Корректное освобождение ресурсов: сервер удаляет все созданные ресурсы (shared memory и семафоры), клиент только закрывает ресурсы, не удаляя их

Код программы

server.c

```
#include <fcntl.h>

#include <stdint.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>
```

```c
#include <unistd.h>
#include <sys/fcntl.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <time.h>

#define SHM_SIZE 4096

int is_prime(int n) {
    if (n <= 1) return 0;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return 0;
    return 1;
}

typedef struct {
    int shm_fd;
    char *shm_buf;
    sem_t *sem_req;
    sem_t *sem_res;
    char shm_name[32];
    char sem_req_name[32];
    char sem_res_name[32];
} shared_resources_t;

void cleanup_resources(shared_resources_t *res) {
    if (res->sem_req != SEM_FAILED) {
        sem_close(res->sem_req);
        sem_unlink(res->sem_req_name);
    }
    if (res->sem_res != SEM_FAILED) {
        sem_close(res->sem_res);
```

```
        sem_unlink(res->sem_res_name);
    }
    if (res->shm_buf != MAP_FAILED) {
        munmap(res->shm_buf, SHM_SIZE);
    }
    if (res->shm_fd != -1) {
        close(res->shm_fd);
        shm_unlink(res->shm_name);
    }
}

int main(int argc, char *argv[])
{
    if (argc < 2) {
        const char msg[] = "Usage: server <filename>\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    char* filename = argv[1];
    shared_resources_t res = {0};
    pid_t client = -1;

    pid_t server_pid = getpid();
    snprintf(res.shm_name, sizeof(res.shm_name), "/prime-shm-%d", server_pid);
    snprintf(res.sem_req_name, sizeof(res.sem_req_name), "/prime-req-%d", server_pid);
    snprintf(res.sem_res_name, sizeof(res.sem_res_name), "/prime-res-%d", server_pid);

    res.shm_fd = shm_open(res.shm_name, O_RDWR | O_CREAT | O_TRUNC, 0600);
    if (res.shm_fd == -1) {
        perror("ERROR: shm_open failed");
        cleanup_resources(&res);
        _exit(EXIT_FAILURE);
```

```
    }

    if (ftruncate(res.shm_fd, SHM_SIZE) == -1) {
        perror("ERROR: ftruncate failed");
        cleanup_resources(&res);
        _exit(EXIT_FAILURE);
    }

    res.shm_buf = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, res.shm_fd,
0);
    if (res.shm_buf == MAP_FAILED) {
        perror("ERROR: mmap failed");
        cleanup_resources(&res);
        _exit(EXIT_FAILURE);
    }

    res.sem_req = sem_open(res.sem_req_name, O_CREAT | O_TRUNC, 0600, 1);
    res.sem_res = sem_open(res.sem_res_name, O_CREAT | O_TRUNC, 0600, 0);
    if (res.sem_req == SEM_FAILED || res.sem_res == SEM_FAILED) {
        perror("ERROR: sem_open failed");
        cleanup_resources(&res);
        _exit(EXIT_FAILURE);
    }

    client = fork();
    if (client == 0) {
        char server_pid_str[32];
        snprintf(server_pid_str, sizeof(server_pid_str), "%d", server_pid);
        char *args[] = {"./client", filename, server_pid_str, NULL};
        execv("./client", args);
        perror("ERROR: execv failed");
        _exit(EXIT_FAILURE);
    } else if (client == -1) {
```

```
        perror("ERROR: fork failed");
        cleanup_resources(&res);
        _exit(EXIT_FAILURE);
    }

    uint32_t *length = (uint32_t *)res.shm_buf;
    char *data = res.shm_buf + sizeof(uint32_t);
    bool running = true;

    while (running) {
        if (sem_wait(res.sem_res) == -1) {
            perror("ERROR: sem_wait failed");
            break;
        }

        if (*length == UINT32_MAX) {
            running = false;
        } else if (*length > 0) {
            int num;
            if (sscanf(data, "%d", &num) == 1) {
                if (num < 0 || is_prime(num)) {
                    *length = UINT32_MAX;
                    running = false;
                } else {
                    char response[32];
                    int response_len = snprintf(response, sizeof(response), "%d\n",
num);
                    if (response_len > (SHM_SIZE - sizeof(uint32_t))) {
                        response_len = SHM_SIZE - sizeof(uint32_t);
                    }
                    *length = response_len;
                    memcpy(data, response, response_len);
                }
```

```
            } else {
                *length = 0;
            }
        }

        sem_post(res.sem_req);

        if (!running) {
            break;
        }
    }

    waitpid(client, NULL, 0);
    cleanup_resources(&res);
    return 0;
}
```

client.c

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <errno.h>
#include <time.h>

#define SHM_SIZE 4096
```

```
int main(int argc, char *argv[])
{
    if (argc < 3) {
        const char msg[] = "Usage: client <filename> <server_pid>\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    char* filename = argv[1];
    int server_pid = atoi(argv[2]);

    char shm_name[32], sem_req_name[32], sem_res_name[32];
    snprintf(shm_name, sizeof(shm_name), "/prime-shm-%d", server_pid);
    snprintf(sem_req_name, sizeof(sem_req_name), "/prime-req-%d", server_pid);
    snprintf(sem_res_name, sizeof(sem_res_name), "/prime-res-%d", server_pid);

    int shm = shm_open(shm_name, O_RDWR, 0);
    if (shm == -1) {
        const char msg[] = "error: failed to open SHM\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    char *shm_buf = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm, 0);
    if (shm_buf == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    sem_t *sem_req = sem_open(sem_req_name, O_RDWR);
    sem_t *sem_res = sem_open(sem_res_name, O_RDWR);
```

```
if (sem_req == SEM_FAILED || sem_res == SEM_FAILED) {
    const char msg[] = "error: failed to open semaphores\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    _exit(EXIT_FAILURE);
}

FILE *file = fopen(filename, "r");
if (!file) {
    const char msg[] = "error: cannot open file\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    _exit(EXIT_FAILURE);
}

uint32_t *length = (uint32_t *)shm_buf;
char *data = shm_buf + sizeof(uint32_t);
bool running = true;
char line[128];

while (running && fgets(line, sizeof(line), file)) {
    sem_wait(sem_req);

    *length = strlen(line);
    memcpy(data, line, *length);

    sem_post(sem_res);

    sem_wait(sem_req);

    if (*length == UINT32_MAX) {
        running = false;
    } else if (*length > 0) {
        write(STDOUT_FILENO, data, *length);
        sem_post(sem_req);
```

```
        }


        if (!running) {
            break;
        }
    }


    if (running) {
        sem_wait(sem_req);
        *length = UINT32_MAX;
        sem_post(sem_res);
    }


    fclose(file);
    sem_close(sem_req);
    sem_close(sem_res);
    munmap(shm_buf, SHM_SIZE);
    close(shm);
    return 0;
}
```

Протокол работы программы

Тестирование:

numbers.txt

```
10
15
7
12
```

test1.txt

```
8
9
```

10

15

3

10

test2.txt

3

4

5

test3.txt

4

8

9

10

100

15

16

18

12

1

15

7

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$ ./server test1.txt

8

9

10

15

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$ ./server numbers.txt

10

15

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$ ./server test2.txt

```
suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$ ./server
test3.txt

4

8

9

10

100

15

16

18

12

1

15

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$
```

Strace:

```
suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$ strace -f
./server numbers.txt

execve("./server", ["./server", "numbers.txt"], 0x7ffd5d2920f0 /* 28 vars */) = 0

brk(NULL)                               = 0x55694ed5c000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f1506d15000

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=19963, ...}) = 0

mmap(NULL, 19963, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1506d10000

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\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\0\0"..., 784, 64) =
784
```

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1506afe000

mmap(0x7f1506b26000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f1506b26000

mmap(0x7f1506cae000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f1506cae000

mmap(0x7f1506cfd000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f1506cfd000

mmap(0x7f1506d03000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1506d03000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1506afb000

arch_prctl(ARCH_SET_FS, 0x7f1506afb740) = 0

set_tid_address(0x7f1506afba10) = 683

set_robust_list(0x7f1506afba20, 24) = 0

rseq(0x7f1506afc060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f1506cfd000, 16384, PROT_READ) = 0

mprotect(0x55692d4e8000, 4096, PROT_READ) = 0

mprotect(0x7f1506d4d000, 8192, PROT_READ) = 0

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

munmap(0x7f1506d10000, 19963) = 0

getpid() = 683

openat(AT_FDCWD, "/dev/shm/prime-shm-683", O_RDWR|O_CREAT|O_TRUNC|O_NOFOLLOW|O_CLOEXEC, 0600) = 3

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f1506d14000

openat(AT_FDCWD, "/dev/shm/sem.prime-req-683", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\xcf\x94\x6a\x8e\x36\xae\x6e\x68", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.v1DqnN", 0x7ffcf6f32680, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.v1DqnN", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0600) = 4

write(4, "\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, 4, 0) = 0x7f1506d13000

link("/dev/shm/sem.v1DqnN", "/dev/shm/sem.prime-req-683") = 0

fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

getrandom("\x93\xe5\xe5\xb4\x88\xe9\x60\xe7", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55694ed5c000

brk(0x55694ed7d000) = 0x55694ed7d000

unlink("/dev/shm/sem.v1DqnN") = 0

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.prime-res-683", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\x31\xa1\x91\x9e\x43\x08\x73\x31", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.3tM7EI", 0x7ffcf6f32680, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.3tM7EI", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0600) = 4

write(4, "\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, 4, 0) = 0x7f1506d12000

link("/dev/shm/sem.3tM7EI", "/dev/shm/sem.prime-res-683") = 0

fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

unlink("/dev/shm/sem.3tM7EI") = 0

close(4) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 684 attached

, child_tidptr=0x7f1506afba10) = 684

[pid 683] futex(0x7f1506d12000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 684] set_robust_list(0x7f1506afba20, 24) = 0

[pid 684] execve("./client", ["./client", "numbers.txt", "683"], 0x7ffcf6f32c50 /* 28 vars */) = 0

[pid 684] brk(NULL) = 0x55a21ff40000

[pid 684] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7effc32c3000

[pid 684] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 684] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 684] fstat(3, {st_mode=S_IFREG|0644, st_size=19963, ...}) = 0

[pid 684] mmap(NULL, 19963, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7effc32be000

[pid 684] close(3) = 0

[pid 684] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 684] 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

[pid 684] 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 684] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 684] 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 684] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7effc30ac000

[pid 684] mmap(0x7effc30d4000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7effc30d4000

[pid 684] mmap(0x7effc325c000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7effc325c000

[pid 684] mmap(0x7effc32ab000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7effc32ab000

[pid 684] mmap(0x7effc32b1000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7effc32b1000

[pid 684] close(3) = 0

[pid 684] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7effc30a9000

[pid 684] arch_prctl(ARCH_SET_FS, 0x7effc30a9740) = 0

[pid 684] set_tid_address(0x7effc30a9a10) = 684

[pid 684] set_robust_list(0x7effc30a9a20, 24) = 0

[pid 684] rseq(0x7effc30aa060, 0x20, 0, 0x53053053) = 0

[pid 684] mprotect(0x7effc32ab000, 16384, PROT_READ) = 0

[pid 684] mprotect(0x55a21ce30000, 4096, PROT_READ) = 0

[pid 684] mprotect(0x7effc32fb000, 8192, PROT_READ) = 0

[pid 684] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 684] munmap(0x7effc32be000, 19963) = 0

[pid 684] openat(AT_FDCWD, "/dev/shm/prime-shm-683", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

[pid 684] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7effc32c2000

[pid 684] openat(AT_FDCWD, "/dev/shm/sem.prime-req-683", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

[pid 684] fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

[pid 684] getrandom("\x2a\x99\x77\x0e\x55\x3f\x63\x93", 8, GRND_NONBLOCK) = 8

[pid 684] brk(NULL) = 0x55a21ff40000

[pid 684] brk(0x55a21ff61000) = 0x55a21ff61000

[pid 684] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7effc32c1000

[pid 684] close(4) = 0

[pid 684] openat(AT_FDCWD, "/dev/shm/sem.prime-res-683", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

[pid 684] fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

[pid 684] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7effc32c0000

[pid 684] close(4) = 0

[pid 684] openat(AT_FDCWD, "numbers.txt", O_RDONLY) = 4

[pid 684] fstat(4, {st_mode=S_IFREG|0777, st_size=15, ...}) = 0

[pid 684] read(4, "10\r\n15\r\n7\r\n12\r\n", 4096) = 15

[pid 684] futex(0x7effc32c0000, FUTEX_WAKE, 1 <unfinished ...>

[pid 683] <... futex resumed>) = 0

[pid 684] <... futex resumed>) = 1

[pid 683] futex(0x7f1506d12000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 684] write(1, "10\n", 310

) = 3

[pid 684] futex(0x7effc32c0000, FUTEX_WAKE, 1 <unfinished ...>

[pid 683] <... futex resumed>) = 0

[pid 684] <... futex resumed>) = 1

[pid 683] futex(0x7f1506d12000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 684] write(1, "15\n", 315

) = 3

[pid 684] futex(0x7effc32c0000, FUTEX_WAKE, 1 <unfinished ...>

[pid 683] <... futex resumed>) = 0

[pid 684] <... futex resumed>) = 1

```
[pid   683] wait4(684,  <unfinished ...>
[pid   684] close(4)                = 0
[pid   684] munmap(0x7effc32c1000, 32)  = 0
[pid   684] munmap(0x7effc32c0000, 32)  = 0
[pid   684] munmap(0x7effc32c2000, 4096) = 0
[pid   684] close(3)                = 0
[pid   684] exit_group(0)           = ?
[pid   684] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)      = 684
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=684, si_uid=1000, si_status=0,
si_utime=0, si_stime=1 /* 0.01 s */} ---
munmap(0x7f1506d13000, 32)          = 0
unlink("/dev/shm/sem.prime-req-683")    = 0
munmap(0x7f1506d12000, 32)          = 0
unlink("/dev/shm/sem.prime-res-683")    = 0
munmap(0x7f1506d14000, 4096)        = 0
close(3)                        = 0
unlink("/dev/shm/prime-shm-683")      = 0
exit_group(0)                   = ?
+++ exited with 0 +++
suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr3$
```

Вывод

В ходе выполнения лабораторной работы была успешно реализована система межпроцессного взаимодействия с использованием механизмов разделяемой памяти (shared memory) и семафоров (semaphores). Программа состоит из двух процессов - сервера и клиента, которые взаимодействуют через общую область памяти и синхронизируют свою работу с помощью двоичных семафоров.