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-210Б-23

Студент: Горячев А.В.

Преподаватель: Бахарев В.Д.

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

Дата: 08.01.25

Москва, 2024

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

● Управлении процессами в ОС

● Обеспечении обмена данных между процессами посредством shared memory

**Задание**

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 shared memory и memory mapping, для синхронизации чтения и записи из shared memory будем использовать семафор.

5 вариант) Пользователь вводит команды вида: «число <endline>». Далее это число передается отродительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

**Общий метод и алгоритм решения**

**Использованные системные вызовы:**

● pid\_t fork(void); – создает дочерний процесс.

● int close(int fd); – сообщает операционной системе об окончании работы с файловым

дескриптором, и закрывает файл(FD).

● int open(const char \* file, int oflag, …); – используется для открытия файла для чтения, записи

или и того, и другого.

● ssize\_t write(int fd, const void \* buf, size\_t n); – Записывает N байт из буфер(BUF) в файл (FD).

Возвращает количество записанных байт или -1.

● void exit(int status); – выполняет немедленное завершение программы. Все используемые

программой потоки закрываются, и временные файлы удаляются, управление возвращается

ОС или другой программе.

● int execlp(const char \*file, const char \*arg, …); – используется для замены текущего процесса

новым процессом, выполняющим указанный исполняемый файл.

● pid\_t wait(int \* stat\_loc); – используются для ожидания изменения состояния процесса-потомка

вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.

● int shm\_open(const char \*name, int oflag, mode\_t mode); – создает и открывает новый (или

открывает уже существующий) объект разделяемой памяти POSIX.

● int shm\_unlink(const char \*name); – удаляется имя объекта разделяемой памяти и, как только

все процессы завершили работу с объектом и отменили его распределение, очищают

пространство и уничтожают связанную с ним область памяти.

● int ftruncate(int fd, off\_t length); – устанавливают длину файла с файловым дескриптором fd в

length байт.

● void \* mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset); – отражает length

байтов, начиная со смещения offset файла (или другого объекта), определенного файловым

дескриптором fd, в память, начиная с адреса start.

● int munmap(void \*start, size\_t length); – удаляет все отражения из заданной области памяти,

после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к

памяти".

● sem\_t \*sem\_open(const char \*name, int oflag); ИЛИ sem\_t \*sem\_open(const char \*name, int oflag,

mode\_t mode, unsigned int value); – создаёт новый семафор или открывает уже существующий.

● int sem\_wait(sem\_t \*sem); – уменьшает значение семафора на 1. Если семафор в данный

момент имеет нулевое значение, то вызов блокируется до тех пор, пока либо не станет

возможным выполнить уменьшение.

● int sem\_post(sem\_t \*sem); – увеличивает значение семафора на 1.

● int sem\_unlink(const char \*name); – удаляет имя семафора из системы. После вызова этой

функции другие процессы больше не смогут открыть этот семафор по имени.

● int sem\_close(sem\_t \*sem); – закрывает указанный семафор, освобождая ресурсы, связанные с

ним.

Родительский процесс создает сегмент общей памяти и инициализирует семафоры для синхронизации. Родительский процесс запрашивает у пользователя имя файла для чтения. Родительский процесс создает дочерний процесс с помощью fork(). В дочернем процессе выполняется программа child.out с использованием execl(), передавая имя файла.

Родительский процесс считывает данные – число – из стандартного ввода, записывает его в общую память, сигнализирует процессу ребенка начать обрабатывать число и ждет сигнала от ребенка.

Дочерний процесс, приняв сигнал, начинает обрабатывать число: если оно составное и положительное, то записывает его файл, сигнализирует родителю о том, что необходимо перейти к следующей итерации принять число от пользователя и ждет до следующего сигнала от родителя. В противном случае дочерний процесс не записывает число в файл, а сигнализирует родителю о том, что необходимо прекратить работу всей программы, и сам завершает свою работу.

Родительский процесс, получив сигнал от ребенка, либо переходит к следующей итерации, либо завершает свою работу.

**Код программы**

**communication.h:**

#ifndef sh

#define sh

#define SHM\_NAME "/shm"

#define REQ\_SEM\_NAME "/req\_sem"

#define RES\_SEM\_NAME "/res\_sem"

typedef struct {

    int number;

    int stop\_flag;

} shared\_data\_t;

#endif

**parrent.c:**

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <string.h>

#include <sys/wait.h>

#include "communication.h"

static char CLIENT\_PROGRAM\_NAME[] = "./child.out";

int read\_str(char \*str, size\_t size) {

    ssize\_t status = read(STDIN\_FILENO, str, size - 1);

    if (status > 0) {

        if (str[status - 1] == '\n') {

            str[status - 1] = '\0';

        } else {

            str[status] = '\0';

            char ch;

            while (read(STDIN\_FILENO, &ch, 1) > 0 && ch != '\n');

            return 1;

        }

        return 0;

    }

    return 1;

}

int main() {

    char filename[130];

    {

        const char msg[] = "Enter the name of file: ";

        write(STDIN\_FILENO, msg, sizeof(msg));

    }

    if (read\_str(filename, sizeof(filename)) != 0) {

        const char msg[] = "Failed to read file\n";

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

        exit(EXIT\_FAILURE);

    }

    sem\_unlink(REQ\_SEM\_NAME);

    sem\_unlink(RES\_SEM\_NAME);

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

    if (shm\_fd == -1) {

        const char msg[] = "Error: shm\_open\n";

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

        exit(EXIT\_FAILURE);

    }

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

        const char msg[] = "Error: ftruncate\n";

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

        exit(EXIT\_FAILURE);

    }

    shared\_data\_t \*shared\_data = mmap(NULL, sizeof(shared\_data\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

    if (shared\_data == MAP\_FAILED) {

        const char msg[] = "Error: mmap\n";

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

        exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_to\_child = sem\_open(REQ\_SEM\_NAME, O\_CREAT, 0666, 0);

    sem\_t \*sem\_to\_parent = sem\_open(RES\_SEM\_NAME, O\_CREAT, 0666, 0);

    if (sem\_to\_child == SEM\_FAILED || sem\_to\_parent == SEM\_FAILED) {

        const char msg[] = "Error: sem\_open\n";

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

        exit(EXIT\_FAILURE);

    }

    pid\_t pid = fork();

    if (pid == -1) {

        const char msg[] = "Error: fork\n";

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

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        int status = execl(CLIENT\_PROGRAM\_NAME, CLIENT\_PROGRAM\_NAME, filename, NULL);

        if (status == -1) {

            const char msg[] = "Error: execl\n";

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

            exit(EXIT\_FAILURE);

        }

    }

    {

        const char msg[] = "Enter numbers. If you want to stop, enter the prime number or Ctrl + D:\n";

        write(STDIN\_FILENO, msg, sizeof(msg));

    }

    shared\_data->stop\_flag = 0;

    while (!shared\_data->stop\_flag) {

        char buffer[30];

        ssize\_t bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer));

        if (bytes\_read <= 0) {

            shared\_data->number = -1;

            sem\_post(sem\_to\_child);

            break;

        }

        int number = atoi(buffer);

        shared\_data->number = number;

        sem\_post(sem\_to\_child);

        sem\_wait(sem\_to\_parent);

    }

    pid\_t status = waitpid(pid, NULL, 0);

    if (status == -1) {

        const char msg[] = "Error: waitpid\n";

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

        exit(EXIT\_FAILURE);

    }

    munmap(shared\_data, sizeof(shared\_data\_t));

    sem\_close(sem\_to\_child);

    sem\_close(sem\_to\_parent);

    sem\_unlink(REQ\_SEM\_NAME);

    sem\_unlink(RES\_SEM\_NAME);

    shm\_unlink(SHM\_NAME);

    return 0;

}

**child.c:**

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <stdio.h>

#include <semaphore.h>

#include <string.h>

#include <sys/types.h>

#include "communication.h"

int is\_prime(int number) {

    if (number < 2) return 0;

    for (int i = 2; i \* i <= number; i++) {

        if (number % i == 0) return 0;

    }

    return 1;

}

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

    if (argc < 2) {

        const char msg[] = "At least one argument is required\n";

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

    }

    const char \*output\_file = argv[1];

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

    if (shm\_fd == -1) {

        const char msg[] = "Error: shm\_open\n";

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

        exit(EXIT\_FAILURE);

    }

    shared\_data\_t \*shared\_data = mmap(NULL, sizeof(shared\_data\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

    if (shared\_data == MAP\_FAILED) {

        const char msg[] = "Error: mmap\n";

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

        exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_to\_child = sem\_open(REQ\_SEM\_NAME, 0);

    sem\_t \*sem\_to\_parent = sem\_open(RES\_SEM\_NAME, 0);

    if (sem\_to\_child == SEM\_FAILED || sem\_to\_parent == SEM\_FAILED) {

        const char msg[] = "Error: sem\_open\n";

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

        exit(EXIT\_FAILURE);

    }

    int fd = open(output\_file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

    if (fd == -1) {

        const char msg[] = "Error: open\n";

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

        exit(EXIT\_FAILURE);

    }

    while (1) {

        sem\_wait(sem\_to\_child);

        int number = shared\_data->number;

        if (number < 0 || is\_prime(number)) {

            shared\_data->stop\_flag = 1;

            sem\_post(sem\_to\_parent);

            break;

        }

        char buffer[32];

        snprintf(buffer, sizeof(buffer), "%d\n", number);

        write(fd, buffer, strlen(buffer));

        shared\_data->stop\_flag = 0;

        sem\_post(sem\_to\_parent);

    }

    close(fd);

    munmap(shared\_data, sizeof(shared\_data\_t));

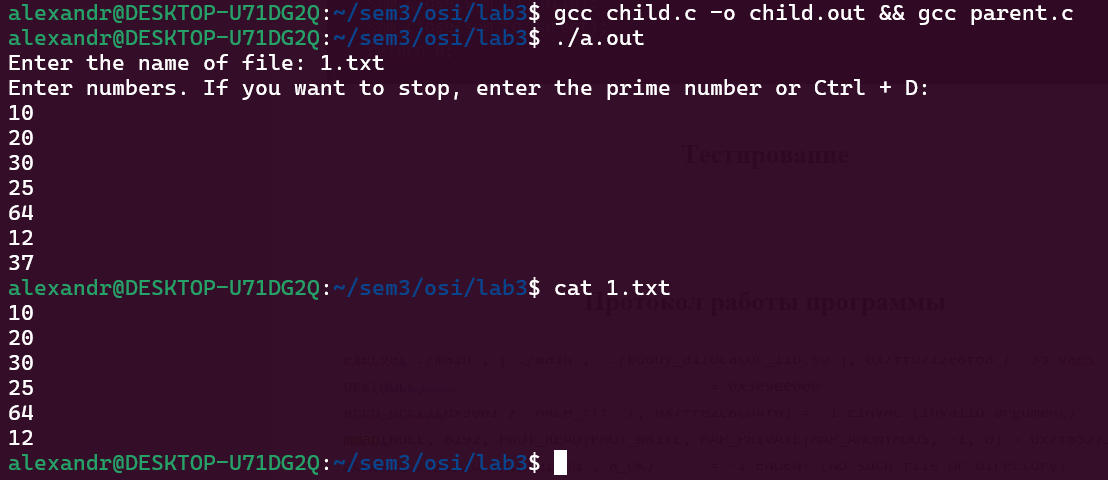
    sem\_close(sem\_to\_child);

    sem\_close(sem\_to\_parent);

    return 0;

}

**Тестирование**



**Протокол работы программы**

strace ./a.out

execve("./a.out", ["./a.out"], 0x7fff67d0c650 /\* 35 vars \*/) = 0

brk(NULL) = 0x2fd5000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe7b5092e0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f2c077ed000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v3/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v3", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v2/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v2", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/x86\_64", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/x86\_64", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64", 0x7ffe7b508500, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}, 0) = 0

openat(AT\_FDCWD, "glibc-hwcaps/x86-64-v3/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "glibc-hwcaps/x86-64-v2/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "libc.so.6", O\_RDONLY|O\_CLOEXEC) = -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=37071, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 37071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f2c077e3000

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\0P\237\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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 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, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f2c075ba000

mprotect(0x7f2c075e2000, 2023424, PROT\_NONE) = 0

mmap(0x7f2c075e2000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f2c075e2000

mmap(0x7f2c07777000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f2c07777000

mmap(0x7f2c077d0000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f2c077d0000

mmap(0x7f2c077d6000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f2c077d6000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f2c075b7000

arch\_prctl(ARCH\_SET\_FS, 0x7f2c075b7740) = 0

set\_tid\_address(0x7f2c075b7a10) = 105875

set\_robust\_list(0x7f2c075b7a20, 24) = 0

rseq(0x7f2c075b80e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f2c077d0000, 16384, PROT\_READ) = 0

mprotect(0x403000, 4096, PROT\_READ) = 0

mprotect(0x7f2c07827000, 8192, PROT\_READ) = 0

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

munmap(0x7f2c077e3000, 37071) = 0

write(0, "Enter the name of file: \0", 25Enter the name of file: ) = 25

read(0, 1.txt

"1.txt\n", 129) = 6

unlink("/dev/shm/sem.req\_sem") = -1 ENOENT (No such file or directory)

unlink("/dev/shm/sem.res\_sem") = -1 ENOENT (No such file or directory)

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

ftruncate(3, 8) = 0

mmap(NULL, 8, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f2c07826000

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

getrandom("\x0a\x79\x0f\x94\xa0\x89\xf0\x54", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.kwRieM", 0x7ffe7b508e30, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.kwRieM", O\_RDWR|O\_CREAT|O\_EXCL, 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) = 0x7f2c077ec000

link("/dev/shm/sem.kwRieM", "/dev/shm/sem.req\_sem") = 0

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

getrandom("\x65\x75\x82\xaf\xc7\x68\x09\x2f", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x2fd5000

brk(0x2ff6000) = 0x2ff6000

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

close(4) = 0

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

getrandom("\x2c\xe9\x25\xdd\x83\xa0\x5b\xc6", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.kBKVyI", 0x7ffe7b508e30, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.kBKVyI", O\_RDWR|O\_CREAT|O\_EXCL, 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) = 0x7f2c077eb000

link("/dev/shm/sem.kBKVyI", "/dev/shm/sem.res\_sem") = 0

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

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

close(4) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f2c075b7a10) = 105912

write(0, "Enter numbers. If you want to st"..., 73Enter numbers. If you want to stop, enter the prime number or Ctrl + D:

) = 73

read(0, 10

"10\n", 30) = 3

futex(0x7f2c077ec000, FUTEX\_WAKE, 1) = 1

futex(0x7f2c077eb000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = -1 EAGAIN (Resource temporarily unavailable)

read(0, 12

"12\n", 30) = 3

futex(0x7f2c077ec000, FUTEX\_WAKE, 1) = 1

futex(0x7f2c077eb000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = -1 EAGAIN (Resource temporarily unavailable)

read(0, 11

"11\n", 30) = 3

futex(0x7f2c077ec000, FUTEX\_WAKE, 1) = 1

wait4(105912, NULL, 0, NULL) = 105912

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=105912, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7f2c07826000, 8) = 0

munmap(0x7f2c077ec000, 32) = 0

munmap(0x7f2c077eb000, 32) = 0

unlink("/dev/shm/sem.req\_sem") = 0

unlink("/dev/shm/sem.res\_sem") = 0

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

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В процессе выполнения данной лабораторной работы я изучил новые системные вызовы на языке Си, которые позволяют эффективно работать с разделяемой памятью и семафорами. Освоил передачу данных между процессами через shared memory и управление доступом с использованием семафоров. Были затруднения на начальном этапе с реализацией задачи с одним семафором: управление процессам передавалось рандомно. Поэтому для упрощения было принято решение использовать два семафора, где все было уже ясно.