

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: Юрченко А.Н.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 23.12.24

Москва, 2024

## Постановка задачи

### Вариант 5.

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:

- `sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value)` - создаёт новый семафор POSIX или открывает существующий семафор.
- `int ftruncate(int fd, off_t length)` - функция `ftruncate` устанавливает длину обычного файла с файловым дескриптором `fd` в `length` байт.
- `void * mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)` - функция `mmap` отражает `length` байтов, начиная со смещения `offset` файла, определенного файловым дескриптором `fd`, в память, начиная с адреса `start`.
- `int shm_open(const char *name, int oflag, mode_t mode)` - создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
- `int sem_wait(sem_t *sem)` - функция уменьшает (блокирует) семафор, на который указывает `sem`.
- `int sem_post(sem_t *sem)` - функция увеличивает (разблокирует) семафор, на который указывает `sem`.
- `int munmap(void *start, size_t length)` - системный вызов `munmap` удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти" (`invalid memory reference`).
- `int shm_unlink(const char *name)` - функция выполняет обратную операцию, удаляя объект, созданный ранее с помощью `shm_open()`.
- `int sem_close(sem_t *sem)` - функция закрывает именованный семафор, на который указывает `sem`, позволяя освободить все ресурсы, которые система выделила под семафор вызывающему процессу.
- `int sem_unlink(const char *name)` - функция удаляет именованный семафор, на который ссылается `name`.
- `int execl(char *name, char *arg0, ... /*NULL*/) - функция (execute) загружает и запускает другую программу. l (список). Аргументы командной строки передаются в форме списка arg0, arg1.... argn, NULL. Эту форму используют, если количество аргументов известно;`

## **parent.c**

**Ввод имени файла:** Программа запрашивает у пользователя имя файла, в который будут записываться составные числа.

### **Создание общей памяти и семафоров:**

- Создается общая память с помощью `shm_open()` и выделяется необходимое пространство с помощью `ftruncate()`.
- Создаются два семафора: `sem_write` (для синхронизации записи в общую память) и `sem_read` (для синхронизации чтения из общей памяти).

### **Создание дочернего процесса:**

- С помощью `fork()` создается дочерний процесс. Если `fork()` успешен, дочерний процесс запускается с помощью `exec()` для выполнения программы `child`.

### **Передача чисел:**

- В родительском процессе в бесконечном цикле запрашиваются числа у пользователя. Пользователь вводит число, которое записывается в общую память.
- Если введенное число отрицательное, то цикл завершается.

## **child.c**

### **Открытие общей памяти и семафоров:**

- Дочерний процесс открывает общую память с помощью `shm_open()` и маппит её в адресное пространство с помощью `mmap()`.
- Открываются семафоры `sem_write` и `sem_read`.

### **Чтение чисел и проверка на простоту:**

- В цикле дочерний процесс ожидает, пока родительский процесс запишет число в общую память. Для этого он использует семафоры: сначала ждет `sem_read`, затем читает число из общей памяти.
- Если число отрицательное, дочерний процесс завершает свою работу.
- Если число составное, оно записывается в файл.

## Код программы

### parent.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <sys/types.h>

#include <sys/wait.h>


#define SHM_NAME "/my_shm"

#define SEM_WRITE_NAME "/sem_write"

#define SEM_READ_NAME "/sem_read"

int main() {

    char filename[256];

    write(STDOUT_FILENO, "Enter the filename to store composite numbers: ", 47);

    int len = read(STDIN_FILENO, filename, sizeof(filename));

    if (len <= 1) {

        const char msg[] = "error: invalid filename\n";

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

        exit(EXIT_FAILURE);

    }

    filename[len - 1] = '\0';

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

    ftruncate(shm_fd, sizeof(int) * 256);
```

```
int *shared_memory = mmap(0, sizeof(int) * 256, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
```

```
sem_t *sem_write = sem_open(SEM_WRITE_NAME, O_CREAT, 0666, 1);
```

```
sem_t *sem_read = sem_open(SEM_READ_NAME, O_CREAT, 0666, 0);
```

```
pid_t pid = fork();
```

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

```
    const char msg[] = "error: fork failed\n";
```

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

```
    exit(EXIT_FAILURE);
```

```
}
```

```
if (pid == 0) {
```

```
    // Дочерний процесс
```

```
    execl("./child", "./child", filename, NULL);
```

```
    _exit(1);
```

```
} else {
```

```
    // Родительский процесс
```

```
    int number;
```

```
    while (1) {
```

```
        write(STDOUT_FILENO, "Enter a number (negative to exit): ", 35);
```

```
        char buffer[256];
```

```
        int len = read(STDIN_FILENO, buffer, sizeof(buffer));
```

```
        if (len <= 1) break;
```

```
        buffer[len - 1] = '\0';
```

```
        number = atoi(buffer);
```

```
        if (number < 0) {
```

```
            sem_wait(sem_write);
```

```
            shared_memory[0] = number;
```

```
            sem_post(sem_read);
```

```

        break;
    }

    sem_wait(sem_write);

    shared_memory[0] = number;

    sem_post(sem_read);
}

wait(NULL);


munmap(shared_memory, sizeof(int) * 256);

shm_unlink(SHM_NAME);

sem_close(sem_write);

sem_close(sem_read);

sem_unlink(SEM_WRITE_NAME);

sem_unlink(SEM_READ_NAME);

write(STDOUT_FILENO, "Parent process exiting.\n", 24);

}

return 0;

}

```

child.c

```

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <stdbool.h>

#include <semaphore.h>

```

```

#define SHM_NAME "/my_shm"

```

```
#define SEM_WRITE_NAME "/sem_write"
```

```
#define SEM_READ_NAME "/sem_read"
```

```
bool is_prime(int num) {
```

```
    if (num <= 1) return false;
```

```
    for (int i = 2; i * i <= num; i++) {
```

```
        if (num % i == 0) return false;
```

```
    }
```

```
    return true;
```

```
}
```

```
int main(int argc, char *argv[]) {
```

```
    if (argc < 2) {
```

```
        const char msg[] = "error: not enough arg\n";
```

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

```
        exit(EXIT_FAILURE);
```

```
    }
```

```
    const char *filename = argv[1];
```

```
    int shm_fd = shm_open(SHM_NAME, O_RDONLY, 0666);
```

```
    int *shared_memory = mmap(0, sizeof(int) * 256, PROT_READ, MAP_SHARED, shm_fd, 0);
```

```
    sem_t *sem_write = sem_open(SEM_WRITE_NAME, 0);
```

```
    sem_t *sem_read = sem_open(SEM_READ_NAME, 0);
```

```
    int file_fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
```

```
    if (file_fd < 0) {
```

```
        const char msg[] = "error with open requested file\n";
```

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

```
        exit(EXIT_FAILURE);
```

```
    }
```

```
int number;

while (1) {

    sem_wait(sem_read);

    number = shared_memory[0];

    if (number < 0) {

        write(STDOUT_FILENO, "Child indicated to terminate\n", 29);

        break;

    }

    if (!(is_prime(number) || number == 1 || number == 0)) {

        char buffer[20];

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

        write(file_fd, buffer, len);

    }

    sem_post(sem_write);

}

close(file_fd);

munmap(shared_memory, sizeof(int) * 256);

shm_unlink(SHM_NAME);

sem_close(sem_write);

sem_close(sem_read);

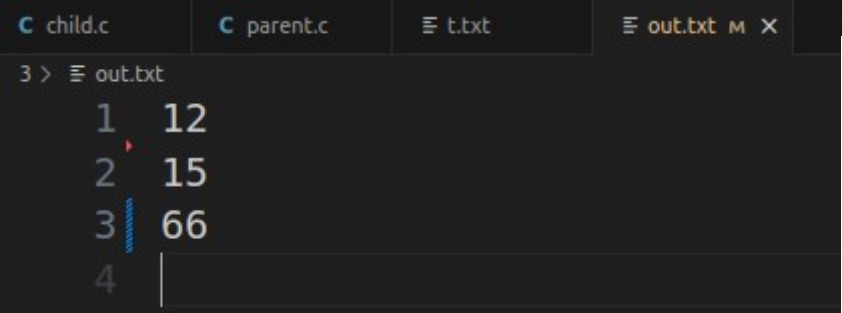

return 0;
```



## Протокол работы программы

### Тестирование:

```
ann@ann-ThinkPad-T460:~/Desktop/osi/3$ gcc -o parent parent.c -lrt -lpthread
ann@ann-ThinkPad-T460:~/Desktop/osi/3$ gcc -o child child.c -lrt -lpthread
ann@ann-ThinkPad-T460:~/Desktop/osi/3$ ./parent
Enter the filename to store composite numbers: out.txt
Enter a number (negative to exit): 1
Enter a number (negative to exit): 0
Enter a number (negative to exit): 2
Enter a number (negative to exit): 12
Enter a number (negative to exit): 15
Enter a number (negative to exit): 66
Enter a number (negative to exit): -1
Child indicated to terminate
Parent process exiting.
```



### Strace:

```
execve("./parent", [ "./parent" ], 0x7ffd3dbbb180 /* 70 vars */) = 0
brk(NULL)                               = 0x6048d05b1000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff3943a2c0) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x726dd14a5000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=66559, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 66559, PROT_READ, MAP_PRIVATE, 3, 0) = 0x726dd1494000
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\3\0>\0\1\0\0\0P\237\2\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"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\1\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) = 0x726dd1200000
```

[illegible]

```
getrandom("\xc1\xb2\x95\x7c\x8b\x0c\xc5\xa0", 8, GRND_NONBLOCK) = 8  
brk(NULL) = 0x6048d05b1000  
brk(0x6048d05d2000) = 0x6048d05d2000  
unlink("/dev/shm/sem.hfLRtM") = 0  
close(4) = 0  
openat(AT_FDCWD, "/dev/shm/sem.sem_read", O_RDWR|O_NOFOLLOW) = -1  
ENOENT (Нет такого файла или каталога)  
getrandom("\x9f\xd2\x8b\x66\xe2\xfb\xcb\xba", 8, GRND_NONBLOCK) = 8  
newfstatat(AT_FDCWD, "/dev/shm/sem.zzW1qr", 0x7fff39439dd0,  
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)  
openat(AT_FDCWD, "/dev/shm/sem.zzW1qr", O_RDWR|O_CREAT|O_EXCL, 0666) = 4  
write(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\0\0", 32) = 32  
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x726dd14a3000  
link("/dev/shm/sem.zzW1qr", "/dev/shm/sem.sem_read") = 0  
newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0  
unlink("/dev/shm/sem.zzW1qr") = 0  
close(4) = 0  
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|  
SIGCHLD, child_tidptr=0x726dd1491a10) = 10325  
write(1, "Enter a number (negative to exit)..., 35Enter a number (negative to exit): ) = 35  
read(0, 2  
"2\n", 256) = 2  
futexp(0x726dd14a3000, FUTEX_WAKE, 1) = 1  
write(1, "Enter a number (negative to exit)..., 35Enter a number (negative to exit): ) = 35  
read(0, 0  
"0\n", 256) = 2  
futexp(0x726dd14a3000, FUTEX_WAKE, 1) = 1  
write(1, "Enter a number (negative to exit)..., 35Enter a number (negative to exit): ) = 35  
read(0, 12  
"12\n", 256) = 3  
futexp(0x726dd14a3000, FUTEX_WAKE, 1) = 1  
write(1, "Enter a number (negative to exit)..., 35Enter a number (negative to exit): ) = 35  
read(0, 15  
"15\n", 256) = 3  
futexp(0x726dd14a3000, FUTEX_WAKE, 1) = 1  
write(1, "Enter a number (negative to exit)..., 35Enter a number (negative to exit): ) = 35  
read(0, 66  
"66\n", 256) = 3  
futexp(0x726dd14a3000, FUTEX_WAKE, 1) = 1  
write(1, "Enter a number (negative to exit)..., 35Enter a number (negative to exit): ) = 35  
read(0, -1
```

```

"-1\n", 256)          = 3
futex(0x726dd14a3000, FUTEX_WAKE, 1Child indicated to terminate
) = 1
wait4(-1, NULL, 0, NULL)      = 10325
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=10325, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
munmap(0x726dd14de000, 1024)   = 0
unlink("/dev/shm/my_shm")      = -1 ENOENT (Нет такого файла или каталога)
munmap(0x726dd14a4000, 32)     = 0
munmap(0x726dd14a3000, 32)     = 0
unlink("/dev/shm/sem.sem_write") = 0
unlink("/dev/shm/sem.sem_read") = 0
write(1, "Parent process exiting.\n", 24Parent process exiting.
) = 24
exit_group(0)                = ?
+++ exited with 0 +++

```

## Вывод

В процессе выполнения этой лабораторной работы я освоила работу с новыми системными вызовами в Си, нужными для взаимодействия с семафорами и разделяемой памятью. Было сложно разобрать во всех функциях и понять как они работают друг и другом. А так работа была интересной.