

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-212БВ-24

Студент: Федосов Д.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 1.11.25

Москва, 2025

## **Постановка задачи**

### **Вариант 18.**

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Необходимо использовать shared memory и memory mapping. Для синхронизации чтения и записи из shared memory использовать семафор.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

## **Общий метод и алгоритм решения**

Использованные системные вызовы:

- `shm_open()` и `ftruncate()`: Создание и установка размера общей памяти.
- `mmap()`: Отображение SHM в виртуальное адресное пространство процесса.
- `sem_open()`: Создание и инициализация семафоров.
- `sem_wait()` и `sem_post()`: Синхронизация доступа в цикле.
- `shm_unlink()` и `sem_unlink()`: Очистка после работы.

Алгоритм решения:

Общий принцип

- Родительский процесс ждет, пока буфер будет пустым, затем записывает данные и сигнализирует, что буфер полон.
- Дочерний процесс ждет, пока буфер будет полон, затем читает данные, обрабатывает их и сигнализирует, что буфер пуст.

Родительский процесс:

- Считывает строку со стандартного ввода.
- Определяет канал (Child 1 или Child 2) на основе счетчика строк (чет/нечет).
- Блокируется вызовом `sem_wait(semEmpty)`, ожидая, пока дочерний процесс подтвердит, что соответствующий буфер свободен для записи.
- Записывает длину строки (`length`) и саму строку (`data`) в Shared Memory.
- Разблокирует дочерний процесс вызовом `sem_post(semFull)`, сигнализируя, что буфер содержит новые данные.
- Повторяет цикл для следующей строки.
- После завершения ввода, ждет, пока оба дочерних процесса не обработают свои последние данные.

- Записывает length = 0 в Shared Memory обоих каналов и сигнализирует, чтобы оба дочерних процесса вышли из цикла ожидания и завершились.
- Ожидает завершения дочерних процессов и очищает все ресурсы.

Дочерний процесс:

- Блокируется вызовом sem\_wait(semFull), ожидая, пока родительский процесс сигнализирует, что буфер содержит данные.
- Считывает длину (length) из Shared Memory.
- Если length == 0, сигнализирует родителю и завершает работу.
- Считывает данные (data) из Shared Memory в объеме, указанном в length.
- Выполняет обработку (удаление гласных) и записывает результат в выходной файл.
- Разблокирует родительский процесс, сигнализируя, что буфер снова свободен для записи.
- Повторяет цикл.
- Закрывает семафоры, отменяет отображение памяти и закрывает файл.

## Код программы

### parent.c

```
#include <fcntl.h>
#include <stdint.h>
#include <stdbool.h>

#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>

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

#define SHM_SIZE 4096
```

```

char SHM_NAME_1[] = "shm1";
char SHM_NAME_2[] = "shm2";

char SEM_EMPTY_NAME_1[] = "semEmpty1";
char SEM_FULL_NAME_1[] = "semFull1";
char SEM_EMPTY_NAME_2[] = "semEmpty2";
char SEM_FULL_NAME_2[] = "semFull2";

typedef struct {
    uint32_t length;
    char data[SHM_SIZE - sizeof(uint32_t)];
} shmStruct;

static char CHILD1_PROGRAM_NAME[] = "child1";
static char CHILD2_PROGRAM_NAME[] = "child2";

void *shmBuf_1 = NULL;
void *shmBuf_2 = NULL;
sem_t *sem_empty_1 = NULL;
sem_t *sem_full_1 = NULL;
sem_t *sem_empty_2 = NULL;
sem_t *sem_full_2 = NULL;

void clear();

short int createChannel(char *shmName, char *semEmptyName, char *semFullName, void **shmBuf, sem_t **semEmpty, sem_t **semFull);

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

    // Получение директории файла

    char progpath[1024];

```

```
{  
    ssize_t len = readlink("/proc/self/exe", progpath,  
                           sizeof(progpath) - 1);  
  
    if (len == -1) {  
  
        const char msg[] = "Ошибка: не удалось прочитать  
полный путь до программы.\n";  
  
        write(STDERR_FILENO, msg, sizeof(msg));  
  
        exit(EXIT_FAILURE);  
    }  
  
    while (progpath[len] != '/')  
        --len;  
  
    progpath[len] = '\0';  
}  
  
// Считывание имён для файлов  
  
char filePath1[512];  
char filePath2[512];  
  
ssize_t bytes1 = read(STDIN_FILENO, filePath1,  
sizeof(filePath1) - 1);  
  
if (bytes1 > 0) {  
  
    if (filePath1[bytes1 - 1] == '\n') {  
  
        filePath1[bytes1 - 1] = '\0';  
  
        bytes1--;  
    } else {  
  
        filePath1[bytes1] = '\0';  
    }  
}  
  
} else {  
  
    write(STDERR_FILENO, "Ошибка ввода имени файла 1.\n",  
50);
```

```

        exit(EXIT_FAILURE);

    }

    ssize_t bytes2 = read(STDIN_FILENO, filePath2,
sizeof(filePath2) - 1);

    if (bytes2 > 0) {

        if (filePath2[bytes2 - 1] == '\n') {

            filePath2[bytes2 - 1] = '\0';

            bytes2--;
        } else {

            filePath2[bytes2] = '\0';

        }
    } else {

        write(STDERR_FILENO, "Ошибка ввода имени файла 2.\n",
50);

        exit(EXIT_FAILURE);
    }
}

// Создание каналов - по SHM и 2 SEM для каждого

if (createChannel(SHM_NAME_1, SEM_EMPTY_NAME_1,
SEM_FULL_NAME_1, &shmBuf_1, &sem_empty_1, &sem_full_1) != 0
|| createChannel(SHM_NAME_2, SEM_EMPTY_NAME_2,
SEM_FULL_NAME_2, &shmBuf_2, &sem_empty_2, &sem_full_2) != 0) {

    clear();

    exit(EXIT_FAILURE);
}

// Вилка child1

pid_t child1 = fork();

if (child1 == 0) {
    char path1[1024];

```

```
    sprintf(path1, sizeof(path1), "%s/%s", progpath,
CHILD1_PROGRAM_NAME);

    char *const args[] = {CHILD1_PROGRAM_NAME, filePath1,
SHM_NAME_1, SEM_FULL_NAME_1, SEM_EMPTY_NAME_1, NULL};

    execv(path1, args);

const char msg[] = "Ошибка: не удалось exec child1.\n";
write(STDERR_FILENO, msg, sizeof(msg));

clear();
exit(EXIT_FAILURE);

} else if (child1 == -1) {

const char msg[] = "Ошибка: не удалось fork child1.\n";
write(STDERR_FILENO, msg, sizeof(msg));
clear();
exit(EXIT_FAILURE);

}

// Вилка child2

pid_t child2 = fork();

if (child2 == 0) {

char path2[1024];

sprintf(path2, sizeof(path2), "%s/%s", progpath,
CHILD2_PROGRAM_NAME);

char *const args[] = {CHILD2_PROGRAM_NAME, filePath2,
SHM_NAME_2, SEM_FULL_NAME_2, SEM_EMPTY_NAME_2, NULL};

execv(path2, args);

const char msg[] = "Ошибка: не удалось exec child2.\n";
write(STDERR_FILENO, msg, sizeof(msg));
```

```
    clear();

    exit(EXIT_FAILURE);

} else if (child2 == -1) {

    const char msg[] = "Ошибка: не удалось fork child2.\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    clear();
    exit(EXIT_FAILURE);
}

// Чтение input'a и запись в SHM

char buf[SHM_SIZE];
size_t counter = 0;
ssize_t bytes;

while (1) {

    bytes = read(STDIN_FILENO, buf, sizeof(buf));

    if (bytes < 0) {
        perror("Ошибка: не удалось получить input.\n");
        clear();
        exit(EXIT_FAILURE);
    }

    if (bytes == 0) {
        break;
    }

    if (bytes > 0 && buf[bytes - 1] == '\n') {
        bytes--;
    }
}
```

```
    if (bytes == 0) {
        break;
    }

    counter++;

    shmStruct *currShm;
    sem_t *currSemEmpty;
    sem_t *currSemFull;

    if (counter % 2 != 0) {
        currShm = (shmStruct *)shmBuf_1;
        currSemEmpty = sem_empty_1;
        currSemFull = sem_full_1;
    } else {
        currShm = (shmStruct *)shmBuf_2;
        currSemEmpty = sem_empty_2;
        currSemFull = sem_full_2;
    }

    sem_wait(currSemEmpty);

    currShm->length = (uint32_t)bytes;
    memcpy(currShm->data, buf, bytes);

    sem_post(currSemFull);
}

if (sem_wait(sem_empty_1) != -1) {
    ((shmStruct *)shmBuf_1)->length = 0;
    sem_post(sem_full_1);
}
```

```

if (sem_wait(sem_empty_2) != -1) {
    ((shmStruct *)shmBuf_2)->length = 0;
    sem_post(sem_full_2);
}

waitpid(child1, NULL, 0);
waitpid(child2, NULL, 0);

clear();
}

// ФУНКЦИЯ создания канала - настройки SHM и SEM

short int createChannel(char *shmName, char *semEmptyName, char
*semFullName, void **shmBuf, sem_t **semEmpty, sem_t **semFull)
{
    int shm;

    shm = shm_open(shmName, O_RDWR | O_CREAT | O_TRUNC, 0600);
    if (shm == -1) {
        const char msg[] = "Ошибка: не удалось создать SHM.\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        return -1;
    }

    if (ftruncate(shm, SHM_SIZE) == -1) {
        const char msg[] = "Ошибка: не удалось установить размер
SHM.\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        close(shm);
        return -1;
    }
}

```

```

        *shmBuf = mmap(NULL, SHM_SIZE, PROT_WRITE, MAP_SHARED, shm,
0);

        if (*shmBuf == MAP_FAILED) {
            const char msg[] = "Ошибка: не удалось отобразить SHM\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            close(shm);
            return -1;
        }

// EMPTY-Семафор - parent может писать - 1

        *semEmpty = sem_open(semEmptyName, O_RDWR | O_CREAT |
O_TRUNC, 0600, 1);

        if (*semEmpty == SEM_FAILED) {
            const char msg[] = "Ошибка: не удалось открыть семафор
EMPTY.\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            munmap(*shmBuf, SHM_SIZE);
            close(shm);
            return -1;
        }

// FULL-Семафор - child может читать - 0

        *semFull = sem_open(semFullName, O_RDWR | O_CREAT | O_TRUNC,
0600, 0);

        if (*semFull == SEM_FAILED) {
            const char msg[] = "Ошибка: не удалось открыть семафор
FULL.\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            sem_close(*semEmpty);
            munmap(*shmBuf, SHM_SIZE);
            close(shm);
            return -1;
        }
    }
}

```

```
    close(shm);

    return 0;
}

// ФУНКЦИЯ ОЧИСТКИ

void clear() {
    if (shmBuf_1 != NULL) {
        munmap(shmBuf_1, SHM_SIZE);
    }

    if (shmBuf_2 != NULL) {
        munmap(shmBuf_2, SHM_SIZE);
    }

    if (sem_empty_1 != NULL) {
        sem_close(sem_empty_1);
    }

    if (sem_full_1 != NULL) {
        sem_close(sem_full_1);
    }

    if (sem_empty_2 != NULL) {
        sem_close(sem_empty_2);
    }

    if (sem_full_2 != NULL) {
        sem_close(sem_full_2);
    }

    shm_unlink(SHM_NAME_1);
    shm_unlink(SHM_NAME_2);
    sem_unlink(SEM_EMPTY_NAME_1);
    sem_unlink(SEM_FULL_NAME_1);
    sem_unlink(SEM_EMPTY_NAME_2);
}
```

```
sem_unlink(SEM_FULL_NAME_2);

return;
}
```

### child1.c и child2.c

```
#include <stdint.h>
#include <stdbool.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <unistd.h>
#include <sys/fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <errno.h>

#define SHM_SIZE 4096

typedef struct {
    uint32_t length;
    char data[SHM_SIZE - sizeof(uint32_t)];
} shmStruct;

void work(char* inBuf, uint32_t inLen, char* outBuf, uint32_t* outLen) {

    const char banwords[] = "aeiouyAEIOUY";
    size_t counter = 0;
```

```
for (size_t i = 0; i < inLen; i++) {
    if (strchr(banwords, inBuf[i]) == NULL) {
        outBuf[counter++] = inBuf[i];
    }
}

*outLen = counter;
}

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

    char *filePath = argv[1];
    char *shmName = argv[2];
    char *semFullName = argv[3];
    char *semEmptyName = argv[4];

    int32_t file = open(filePath, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1) {
        perror("Ошибка: не удалось открыть файл.");
        _exit(EXIT_FAILURE);
    }

    int shm = shm_open(shmName, O_RDWR, 0600);
    if (shm == -1) {
        perror("Ошибка: не удалось открыть SHM.");
        close(file);
        _exit(EXIT_FAILURE);
    }

    shmStruct *shmBuf = (shmStruct *)mmap(NULL, SHM_SIZE,
PROT_READ | PROT_WRITE, MAP_SHARED, shm, 0);
```

```
close(shm);

if (shmBuf == MAP_FAILED) {
    perror("Ошибка: не удалось отобразить SHM.");
    close(file);
    _exit(EXIT_FAILURE);
}

sem_t *semFull = sem_open(semFullName, O_RDWR);
if (semFull == SEM_FAILED) {
    perror("Ошибка: не удалось открыть семафор FULL.");
    munmap(shmBuf, SHM_SIZE);
    close(file);
    _exit(EXIT_FAILURE);
}

sem_t *semEmpty = sem_open(semEmptyName, O_RDWR);
if (semEmpty == SEM_FAILED) {
    perror("Ошибка: не удалось открыть семафор EMPTY.");
    sem_close(semFull);
    munmap(shmBuf, SHM_SIZE);
    close(file);
    _exit(EXIT_FAILURE);
}

char outBuf[SHM_SIZE];
uint32_t outLen;

while (true) {
    sem_wait(semFull);

    uint32_t len = shmBuf->length;
```

```
    if (len == 0) {

        sem_post(semEmpty);
        break;
    }

    work(shmBuf->data, len, outBuf, &outLen);

    if (outLen > 0) {

        write(file, outBuf, outLen);
        write(file, "\n", 1);
    }

    sem_post(semEmpty);
}

sem_close(semFull);
sem_close(semEmpty);
munmap(shmBuf, SHM_SIZE);
close(file);

return EXIT_SUCCESS;
}
```

## Протокол работы программы

```
snadon@fedora:~/OS-Labs/Lab3$ ./parent
1
2
Hola!
I'm Tired
:D
:(()
```

```
|snadon@fedora:~/OS-Labs/Lab3$ 
```

Файл 1:

```
->
1 Hl!
2 :D
3 |
```

Файл 2:

```
=<
1 'm Trd
2 :(()
3
```

## Вывод

В процессе выполнения лабораторной работы я составил программу, подобную программе из лабораторной №1, но при этом осуществляющую работу с shared memory и семафорами. Я приобрел базовые навыки использования shared memory и memory mapping. Одной из основных сложностей была в правильной регулировке выполнения процессов с помощью семафоров.