

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-209БВ-24

Студент: Лисов Д.С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_

Дата: 12.11.25

Москва, 2025

## Постановка задачи

### Вариант 15.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программы (основной процесс) должен создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересыпает их в File Mapping. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в Error File Mapping выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода

## Общий метод и алгоритм решения

Использованные системные вызовы:

- open() - открытие файла;
- ftruncate() - установка размера файла;
- mmap() - отображение файла в память;
- munmap() - удаление отображения;
- fork() - создание нового процесса;
- execv() - замена образа процесса;
- wait() - ожидание завершения процесса;
- kill() - отправка сигналов;
- signal() - обработка сингалов;
- fstat() - получение информации о файле;
- pause() - ожидание сигнала;
- close() - закрытие файлового дескриптора

Алгоритм аналогичен описанному в условии задачи: создаётся File Mapping для обмена строк между процессами и Error File Mapping для переотправки ошибок от дочернего процесса.

## Код программы

### parent.c

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

#define MAX_LEN 4096
```

```

volatile sig_atomic_t child_done = 0;

typedef struct {
    char message[MAX_LEN];
    int has_message;
    int terminate;
} shared_data_t;

void handler_parent(int sig) {
    child_done = 1;
}

int main() {
    char *filename = "FileMapping";

    int fd = open(filename, O_RDWR | O_CREAT, 0666);
    int fd_errors = open("Errors File Mapping", O_CREAT | O_RDWR, 0666);

    if (fd == -1 || fd_errors == -1) {
        perror("open");
        exit(1);
    }

    if (ftruncate(fd, sizeof(shared_data_t)) == -1) {
        perror("ftruncate");
        close(fd);
        exit(1);
    }

    if (ftruncate(fd_errors, sizeof(shared_data_t)) == -1) {
        perror("ftruncate");
        close(fd_errors);
        exit(1);
    }

    shared_data_t *data = mmap(NULL, sizeof(shared_data_t), PROT_READ |
PROT_WRITE, MAP_SHARED, fd, 0);
    shared_data_t *errors = mmap(NULL, sizeof(shared_data_t), PROT_READ |
PROT_WRITE, MAP_SHARED, fd_errors, 0);
    if (data == MAP_FAILED || errors == MAP_FAILED) {
        perror("mmap");
        close(fd);
        exit(1);
    }
}

```

```
data->message[0] = '\0';
data->has_message = 0;
data->terminate = 0;

char *s = NULL;
size_t len = 0;
getline(&s, &len, stdin);

if (s[strlen(s) - 1] == '\n') {
    s[strlen(s) - 1] = '\0';
}
strncpy(data->message, s, 255);

signal(SIGUSR2, handler_parent);
pid_t pid = fork();

if (pid == -1) {
    perror("fork");
    munmap(data, sizeof(shared_data_t));
    close(fd);
    exit(1);
}

if (pid > 0) {
    while (1) {
        fflush(stdout);

        ssize_t read_bytes = getline(&s, &len, stdin);
        if (read_bytes == -1) {
            break;
        }

        if (s[read_bytes - 1] == '\n') {
            s[read_bytes - 1] = '\0';
            --read_bytes;
        }

        if (strcmp(s, "exit") == 0) {
            data->has_message = 1;
            data->terminate = 1;
            kill(pid, SIGUSR1);
            break;
        }
    }
}
```

```

        while (data->has_message) {
            usleep(10000);
        }

        strncpy(data->message, s, MAX_LEN - 1);
        data->message[MAX_LEN - 1] = '\0';
        data->has_message = 1;

        printf("Отправлено через File Mapping: %s\n", s);
        kill(pid, SIGUSR1);

/*while (!errors->has_message && !errors->terminate) {
    usleep(10000);
}*/
child_done = 0;
while (!child_done) {
    pause();
}

if (errors->has_message)
{
>message);           printf("Дочерний процесс ответил: %s\n", errors-
    errors->has_message = 0;
}
errors->terminate = 0;
}

free(s);

wait(NULL);
} else {
    char *args[] = {"./child", filename, NULL};
    execv("./child", args);
    perror("execv");
    exit(1);
}

munmap(data, sizeof(shared_data_t));
close(fd);
return 0;
}

```

### child.c

```

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

```

```
#include <string.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/stat.h>
#include <signal.h>

#define MAX_LEN 4096

volatile sig_atomic_t has_signal = 0;

typedef struct {
    char message[MAX_LEN];
    int has_message;
    int terminate;
} shared_data_t;

void handler_child(int sig) {
    has_signal = 1;
}

int main(int argc, char* argv[]) {
    signal(SIGUSR1, handler_child);
    char *mapping_filename = argv[1];

    char *error_filename = "Errors File Mapping";
    char output_filename[256];

    int fd = open(mapping_filename, O_RDWR);
    int fd_errors = open(error_filename, O_RDWR);

    if (fd == -1 || fd_errors == -1) {
        perror("open");
        exit(1);
    }

    struct stat st;
    fstat(fd, &st);
    if (st.st_size < sizeof(shared_data_t)) {
        fprintf(stderr, "File too small: %ld < %zu\n", st.st_size,
        sizeof(shared_data_t));
        exit(1);
    }
    shared_data_t *shared_data = mmap(NULL, sizeof(shared_data_t),
        PROT_WRITE | PROT_READ, MAP_SHARED, fd, 0);
```

```
| PROT_READ, MAP_SHARED, error_data = mmap(NULL, sizeof(shared_data_t), PROT_WRITE  
  
if (shared_data == MAP_FAILED || error_data == MAP_FAILED) {  
    perror("mmap");  
    close(fd);  
    exit(1);  
}  
  
strncpy(output_filename, shared_data->message, 255);  
  
FILE* output_file = fopen(output_filename, "w");  
  
printf("Открыт файл %s для записи\n", output_filename);  
  
while(1) {  
    while (!has_signal) {  
        pause();  
    }  
  
    has_signal = 0;  
    printf("Получено через File Mapping: %s\n", shared_data->message);  
  
    if (shared_data->terminate) {  
        printf("Получен сигнал завершения\n");  
        break;  
    }  
  
    if (isupper((unsigned char)shared_data->message[0])) {  
        fprintf(output_file, "%s\n", shared_data->message);  
        fflush(output_file);  
        error_data->terminate = 1;  
        error_data->has_message = 0;  
    } else {  
        strncpy(error_data->message, "ERROR!", strlen("ERROR!"));  
        // error_data->terminate = 0;  
        error_data->has_message = 1;  
        kill(getppid(), SIGUSR2);  
    }  
  
    shared_data->has_message = 0;  
  
    kill(getppid(), SIGUSR2);  
    // usleep(10000);  
}
```

```

    fclose(output_file);
    munmap(shared_data, sizeof(shared_data_t));
    munmap(error_data, sizeof(shared_data_t));

    close(fd_errors);
    close(fd);

    return 0;
}

```

## Протокол работы программы

### Тестирование:

```

$ ./parent
example.txt
Открыт файл example.txt для записи
A
Отправлено через File Mapping: A
Получено через File Mapping: A
Aboba
Отправлено через File Mapping: Aboba
Получено через File Mapping: Aboba
E
Отправлено через File Mapping: E
Получено через File Mapping: E
error
Отправлено через File Mapping: error
Получено через File Mapping: error
Дочерний процесс ответил: ERROR!
buffer
Отправлено через File Mapping: buffer
Получено через File Mapping: buffer
Дочерний процесс ответил: ERROR!
exit
Получено через File Mapping: buffer
Получен сигнал завершения

```

### Strace:

```

$ strace -f ./parent
execve("./parent", ["/./parent"], 0x7fff13d0c108 /* 63 vars */) = 0
brk(NULL)                               = 0x63f913080000
 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
каталога) access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=153815, ...}) = 0

```





```
[pid 7743] write(1, "Hello\n", 6) = 6
[pid 7743] kill(7745, SIGUSR1)      = 0
[pid 7745] <... pause resumed>     = ? ERESTARTNOHAND (To be restarted if
no handler)
[pid 7743] pause( <unfinished ...>
si_uid=1000) --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=7743,
[pid 7745] rt_sigreturn({mask=[]})    = -1 EINTR (Прерван системный вызов)
[pid 7745] write(1, "Hello\n", 6) = 6
[pid 7743] <... pause resumed>     = ? ERESTARTNOHAND (To be restarted if
no handler)
[pid 7743] pause( <unfinished ...>
si_uid=1000) --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=7745,
[pid 7745] rt_sigreturn({mask=[]})    = -1 EINTR (Прерван системный вызов)
[pid 7743] read(0, Abo abof
"Abo abof\n", 1024) = 9
[pid 7743] write(1, "Abo abof\n", 9) = 9
[pid 7743] kill(7745, SIGUSR1)      = 0
[pid 7743] pause( <unfinished ...>
no handler)
[pid 7745] <... pause resumed>     = ? ERESTARTNOHAND (To be restarted if
si_uid=1000)
[pid 7745] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=7743,
[pid 7745] rt_sigreturn({mask=[]})    = -1 EINTR (Прерван системный вызов)
[pid 7743] write(1, "Abo abof\n", 9) = 9
[pid 7743] getppid()                = 7743
[pid 7743] kill(7745, SIGUSR2)      = 0
[pid 7743] <... pause resumed>     = ? ERESTARTNOHAND (To be restarted if
no handler)
[pid 7743] pause( <unfinished ...>
si_uid=1000) --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=7745,
[pid 7745] rt_sigreturn({mask=[]})    = -1 EINTR (Прерван системный вызов)
[pid 7743] read(0, hi
"hi\n", 1024) = 3
[pid 7743] write(1, "hi\n", 3) = 3
[pid 7743] kill(7745, SIGUSR1)      = 0
[pid 7743] <... pause resumed>     = ? ERESTARTNOHAND (To be restarted if
no handler)
```

```
[pid  7743] pause( <unfinished ...>
[pid  7745] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=7743,
si_uid=1000}           = -1 EINTR (Прерван системный вызов)
[pid  7745] rt_sigreturn({mask=[]})      = -1 EINTR (Прерван системный вызов)
[pid  7745] write(1, "320\276\321\203\321\207\320\265\320\275\320\276\321\207\320\265\321\200\320\265\320\267 File...", 45Получено через File Mapping: hi
) = 45
[pid  7745] getppid()                  = 7743
[pid  7745] kill(7743, SIGUSR2)        = 0
no handler) <... pause resumed>      = ? ERESTARTNOHAND (To be restarted if
[pid  7745] getppid( <unfinished ...>
[pid  7743] --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=7745,
si_uid=1000}           = -1 EINTR (Прерван системный вызов)
[pid  7745] <... getppid resumed>     = 7743
[pid  7743] rt_sigreturn({mask=[]} <unfinished ...>
[pid  7745] kill(7743, SIGUSR2 <unfinished ...>
[pid  7743] <... rt_sigreturn resumed> = -1 EINTR (Прерван системный вызов)
[pid  7745] <... kill resumed>        = 0
[pid  7743] --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=7745,
si_uid=1000}           = -1 EINTR (Прерван системный вызов)
[pid  7745] pause( <unfinished ...>
[pid  7743] rt_sigreturn({mask=[]})      = -1 EINTR (Прерван системный вызов)
[pid  7745] write(1, "320\276\321\200\320\266\320\265\321\206\320\265\321\201\321\201 File...", 55Дочерний процесс
ответил: ERROR!
) = 55
[pid  7743] read(0, Hewfrwq
"Hewfrwq\n", 1024) = 8
[pid  7743] write(1, "320\276\321\202\321\207\321\200\320\260\320\262\320\273\320\265\320\275\320\276\321\207\320\265\321\200\320\265\320\267 File Mapping: Hewfrwq
) = 54
[pid  7743] kill(7745, SIGUSR1)        = 0
no handler) <... pause resumed>      = ? ERESTARTNOHAND (To be restarted if
[pid  7743] pause( <unfinished ...>
[pid  7743] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=7743,
si_uid=1000}           = -1 EINTR (Прерван системный вызов)
[pid  7743] write(1, "320\276\321\203\321\203\321\207\320\265\320\275\320\276\321\207\320\265\321\200\320\265\320\267 File...", 50Получено через File Mapping:
Hewfrwq
) = 50
[pid  7745] write(7, "Hewfrwq\n", 8)   = 8
[pid  7745] getppid()                  = 7743
[pid  7745] kill(7743, SIGUSR2)        = 0
no handler) <... pause resumed>      = ? ERESTARTNOHAND (To be restarted if
[pid  7745] pause( <unfinished ...>
[pid  7743] --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=7745,
si_uid=1000}           = -1 EINTR (Прерван системный вызов)
[pid  7743] rt_sigreturn({mask=[]})      = -1 EINTR (Прерван системный вызов)
[pid  7743] read(0, exit
"exit\n", 1024)      = 5
[pid  7743] kill(7745, SIGUSR1)        = 0
```

## **Вывод**

Лабораторная работа показалась достаточно интересной и важной, поскольку научился работать с разделяемой памятью и отображением файла в память процесса (File Mapping).

**Освоил системные вызовы mmap, munmap, ftruncate. Реализована синхронизация процессов через сигналы (SIGUSR1, SIGUSR2).**

Сталкивался со сложностью работы со строками произвольной длины, поскольку mmap создаёт фиксированное отображение.