

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Факультет информационных технологий и  
прикладной математики  
Кафедра вычислительной математики и  
программирования

**Лабораторная работа №4 по курсу  
"Операционные системы"**

**MEMORY MAPPING**

Студент: Сеимов Максим Сергеевич

Группа: М8О-208Б-21

Вариант: 9

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_

Дата: \_\_\_\_\_

Подпись: \_\_\_\_\_

# 1 Постановка задачи

## Цель работы

Целью лабораторной работы №4 является приобретение практических навыков в:

- Освоении принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «*File mapping*»

## 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 системные сигналы/события и/или через отображаемые файлы (*memory-mapped files*). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

## Вариант 9

В варианте №9 нужно написать программу, представленную двумя процессами: родительский принимает у пользователя имя файла, создаёт дочерний, который из введённого файла считывает команды вида [число число число\n], и отправляет в родительский процесс результаты деления первого числа на последующие в строке; числа представлены типом `float`.

## Общие сведения о программе

Программа представлена одним исполняемым файлом **lab4**, компилируемым из файла **main.c**. Используются заголовочные файлы **sys/types.h**, **sys/stat.h**, **sys/mman.h**, **fcntl.h**, **wait.h**, **unistd.h**, **stdlib.h**, **stdio.h**, **string.h**. В программе используются следующие системные вызовы:

- **fork** - Создание дочернего процесса (копия родительского с другим идентификатором)
- **execve** - Замена образа памяти процесса другим исполняемым файлом
- **waitpid** - Ожидание завершения дочернего процесса
- **mmap** - Маппинг файла с диска в оперативную память
- **munmap** - Выгрузка файла из памяти
- **exit** - Завершение выполнения процесса и возвращение статуса
- **dup2** - Переназначение файлового дескриптора
- **open** - Открытие или создание файла
- **close** - Заккрытие файла

## Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы системных вызовов **mmap** и **munmap**
2. В коде родительского процесса выделить память с помощью **mmap** и флагами **MAP\_SHARED** и **MAP\_ANONYMOUS** для выделения пустого общего блока. После вызова **fork** в дочерней ветке переназначить файловые дескрипторы файла, открытого на чтение, и записывающего конца неименованного канала как стандартные потоки ввода-вывода. В родительской ветке ждать завершения работы дочернего процесса
3. Написать код дочернего процесса (в том же **main.c** после вызова **fork**), выполняющего простейшие операции деления для команд указанного типа и записывающий результат в выделенную память с отступом в **sizeof(int)** от начала, куда потом в конце он запишет количество чисел в памяти
4. После завершения дочернего процесса вывести в родительском результаты из выделенной памяти считать количество записанных дочерним процессом чисел и вывести их пользователю.

## Основные файлы программы

**main.c:**

```
1 | #include <sys/types.h>
2 | #include <sys/stat.h>
3 | #include <sys/mman.h>
4 | #include <fcntl.h>
5 | #include <wait.h>
6 | #include <unistd.h>
7 | #include <stdlib.h>
8 | #include <stdio.h>
9 | #include <string.h>
10 |
11 | #define STR_LEN 128
12 | #define MAX_LENGTH 100
13 |
14 | int first = 1;
15 | int ind = 0;
16 |
17 | int read_float(int fd, float *f) {
18 |     char *buf = calloc(sizeof(char), STR_LEN);
19 |     char c = 0;
20 |     short i = 0;
21 |
22 |     if (read(fd, &c, 1) < 0) {
23 |         exit(3);
24 |     }
25 |
26 |     if ((c == ' ') || (c == '\n') || (c == '\0'))
27 |         return -1;
28 |
29 |     while (c != ' ' && c != '\n') {
30 |         buf[i++] = c;
31 |         read(fd, &c, 1);
32 |     }
33 | }
```

```

34     *f = strtod(buf, NULL);
35
36     if (c == '\n') {
37         return 0;
38     }
39
40     return 1;
41 }
42
43 int exec_command(float *result) {
44
45     int read_result;
46
47     float init_num = 0, div_num = 0;
48
49     if ((read_result = read_float(0, &init_num)) == -1) {
50
51         if (first == 1)
52             exit(4);
53         else
54             return 0;
55     }
56
57     else if (read_result == 0) {
58         if (ind >= MAX_LENGTH) {
59             exit(6);
60         }
61         result[ind++] = init_num;
62         first = 0;
63     }
64
65     else {
66         first = 0;
67         while ((read_result = read_float(0, &div_num)) != -1) {
68
69             if (div_num == 0) {
70                 exit(1);
71             }
72
73             init_num /= div_num;
74
75             if (read_result == 0) {
76                 break;
77             }
78         }
79         if (ind >= MAX_LENGTH) {
80             exit(6);
81         }
82         result[ind++] = init_num;
83     }
84
85     return 1;
86 }
87
88
89 size_t strlen_new(char *s) {
90     for (size_t i = 0; i < strlen(s); ++i) {
91         if (s[i] == '\n')
92             return i;
93     }

```

```

94     return strlen(s);
95 }
96
97 int main() {
98
99     char *fname_buf = calloc(sizeof(char), STR_LEN);
100     if (read(0, fname_buf, STR_LEN) < 0) {
101         perror("Read error");
102         exit(EXIT_FAILURE);
103     }
104
105     char *fname = malloc(sizeof(char) * strlen_new(fname_buf));
106     strncpy(fname, fname_buf, strlen_new(fname_buf));
107     free(fname_buf);
108
109     void *shared = mmap(NULL, sizeof(int) + sizeof(float) * MAX_LENGTH, PROT_READ |
        PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
110
111     if (shared == MAP_FAILED) {
112         perror("Mapping error");
113         exit(EXIT_FAILURE);
114     }
115
116     int filedес;
117     if ((filedes = open(fname, O_RDONLY)) < 0) {
118         perror(fname);
119         exit(EXIT_FAILURE);
120     }
121
122     pid_t pid = fork();
123
124     if (pid < 0) {
125         perror("Fork error");
126         exit(EXIT_FAILURE);
127     }
128
129     if (pid == 0) { // Child
130
131         float *shared_arr = (float *) (shared + sizeof(int));
132         if (dup2(filedes, 0) < 0) {
133             perror("Descriptor redirection error (input)");
134             exit(EXIT_FAILURE);
135         }
136
137         while (exec_command(shared_arr) == 1) {
138             ;
139         }
140
141         * (int *) shared = ind;
142
143         if (munmap(shared, MAX_LENGTH) < 0) {
144             perror("munmap");
145             exit(EXIT_FAILURE);
146         }
147         if (close(filedes) < 0) {
148             perror("Close error");
149             exit(EXIT_FAILURE);
150         }
151
152         return 0;

```

```

153     }
154
155     else { // Parent
156
157         int child_exit_status;
158         if (waitpid(pid, &child_exit_status, 0) < 0) {
159             perror("Waitpid error");
160         }
161
162         if (WEXITSTATUS(child_exit_status) == 1) {
163             char *err1 = "Child process error exit, divison by zero\n";
164             write(2, err1, strlen(err1));
165             exit(EXIT_FAILURE);
166         }
167
168         else if (WEXITSTATUS(child_exit_status) == 4) {
169             char *err2 = "Child process error exit, expected float number\n";
170             write(2, err2, strlen(err2));
171             exit(EXIT_FAILURE);
172         }
173
174         else if (WEXITSTATUS(child_exit_status) == 6) {
175             char *err3 = "Child process error exit, too many strings\n";
176             write(2, err3, strlen(err3));
177             exit(EXIT_FAILURE);
178         }
179
180         else if (WEXITSTATUS(child_exit_status) != 0) {
181             perror("Child process error");
182             exit(EXIT_FAILURE);
183         }
184
185         int number = * (int *) shared;
186         float result;
187         float *shared_arr = (float *) (shared + sizeof(int));
188
189         for (int i = 0; i < number; ++i) {
190             result = shared_arr[i];
191             char *result_string = calloc(sizeof(char), STR_LEN);
192             gcvt(result, 7, result_string);
193             if (write(1, result_string, strlen(result_string)) < 0) {
194                 perror("Can't write result to stdout");
195                 exit(EXIT_FAILURE);
196             }
197
198             char endl_c = '\n';
199             if (write(1, &endl_c, 1) < 0) {
200                 perror("Can't write \\n to stdout");
201                 exit(EXIT_FAILURE);
202             }
203         }
204
205         if (munmap(shared, MAX_LENGTH) < 0) {
206             perror("Munmap error");
207             exit(EXIT_FAILURE);
208         }
209         if (close(filedes) < 0) {
210             perror("Close error");
211             exit(EXIT_FAILURE);
212         }

```

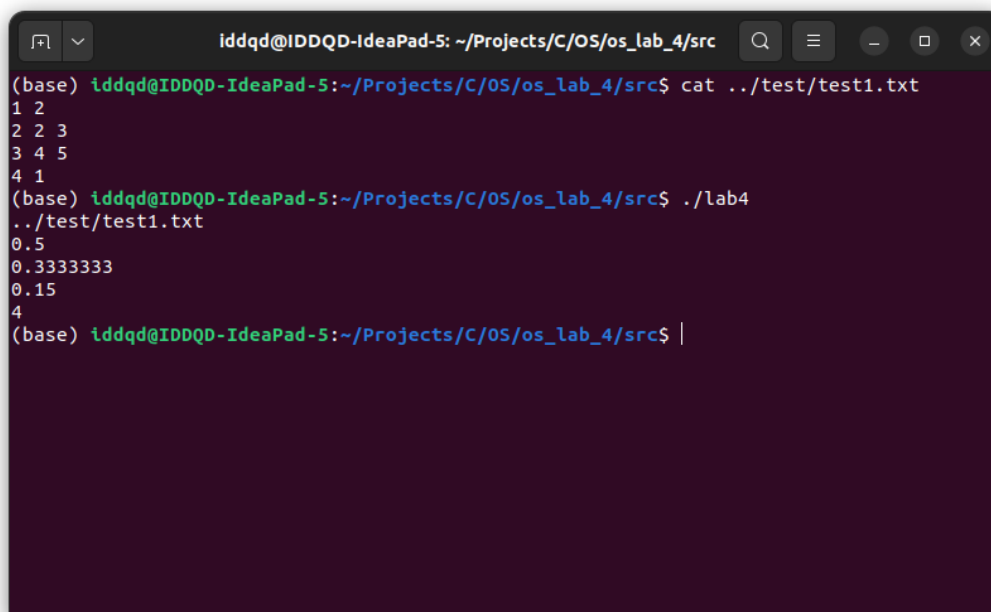
```

213 |
214 |     }
215 |
216 |     return 0;
217 | }

```

## Пример работы

Запуск программы на нескольких тестах (вывод полностью аналогичен ЛР №2):

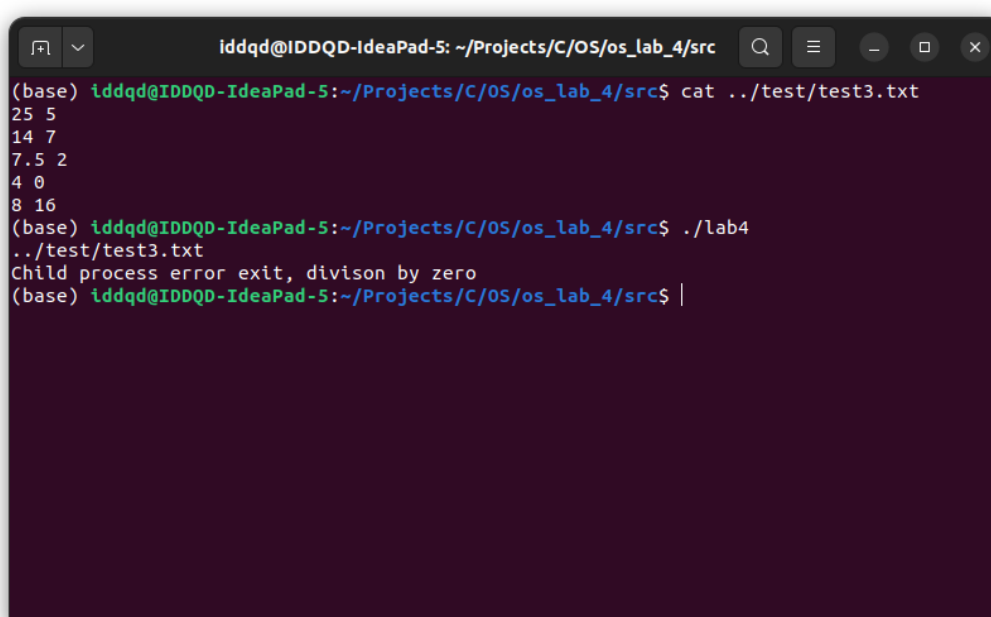


```

(base) iddq@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_4/src$ cat ../test/test1.txt
1 2
2 2 3
3 4 5
4 1
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ ./lab4
../test/test1.txt
0.5
0.3333333
0.15
4
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ |

```

Рис. 1: Простейший пример запуска программы с несколькими командами



```

(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ cat ../test/test3.txt
25 5
14 7
7.5 2
4 0
8 16
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ ./lab4
../test/test3.txt
Child process error exit, division by zero
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ |

```

Рис. 2: При делении на ноль хотя бы в одной команде оба процесса завершаются

```
iddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_4/src
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ cat ../test/test4.txt
25
8
16.92
0
1
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ ./lab4
../test/test4.txt
25
8
16.92
0
1
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ |
```

Рис. 3: Если в команде содержится только одно число, оно и будет результатом выполнения

```
iddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_4/src
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ cat ../test/test6.txt
10.0 1.22 +3.6623 -0.023
0.24 -113 0.0001 -24.5 -3
0 11 92.1 0.221
-3 -0.00000001
4e+03 8
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ ./lab4
../test/test6.txt
-97.3102
-0.2889651
0
3e+08
500
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_4/src$ |
```

Рис. 4: Числа могут записываться как целые или как десятичные дроби, также возможно использование знаков



## Вывод

В ходе выполнения задания были изучены принципы работы системных вызовов `mmap` и `mmap2` как нового способа обеспечения межпроцессорного взаимодействия.