Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени
Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине «Разработка программных систем»

Студент:	Новокшанов Евгений Андреевич
Группа:	РК6-66Б
Тип задания:	лабораторная работа
Тема:	Многопроцессное программирование
Вариант:	3

Студент _____ Новокшанов Е.А.
подпись, дата Фамилия, И.О.

Преподаватель _____ Козов А.В.
подпись, дата Фамилия, И.О.

Москва, 2023

Содержание

Задание

Составить программу параллельного чтения и печати на стандартный вывод содержимого любого текстового файла 2-мя процессами так, чтобы 1-ый процесс преобразовывал перед печатью все прочитанные им буквы в заглавные, а процесс 2 - в строчные. Имя обрабатываемого файла должно передаваться программе как аргумент командной строки.

Внимание! Файл должен открываться посредством open однократно до выполнения системного вызова fork. Чтение файла каждым процессом должно выполняться побайтно с помощью read с небольшой задержкой (см. usleep()).

Описание структуры программы и реализованных способов взаимодействия процессов

Программный код можно разделить на 2 составные части: Проверка входных данных: Проверка аргументов командной строки и открытие файла. В случае ошибки будет выведено соответствующее сообщение:

- «Error: Restart and input one argument(filename)»;
- «Error: can't open file».

Работа с процессами: Создание дочернего процесса системным вызовом **fork()**. Далее 2 процесса, процесс-отец и процесс-сын, выполняются параллельно: посимвольно считывают данные из файла, имя которого мы ввели в виде аргумента командной строки. Процесс-сын преобразует приводит прочитанный символ к нижнему регистру и выводит его в стандартный поток вывода(1). Процесс-отец выполняет те же действия, но выводит символы приведенные к верхнему регистру.

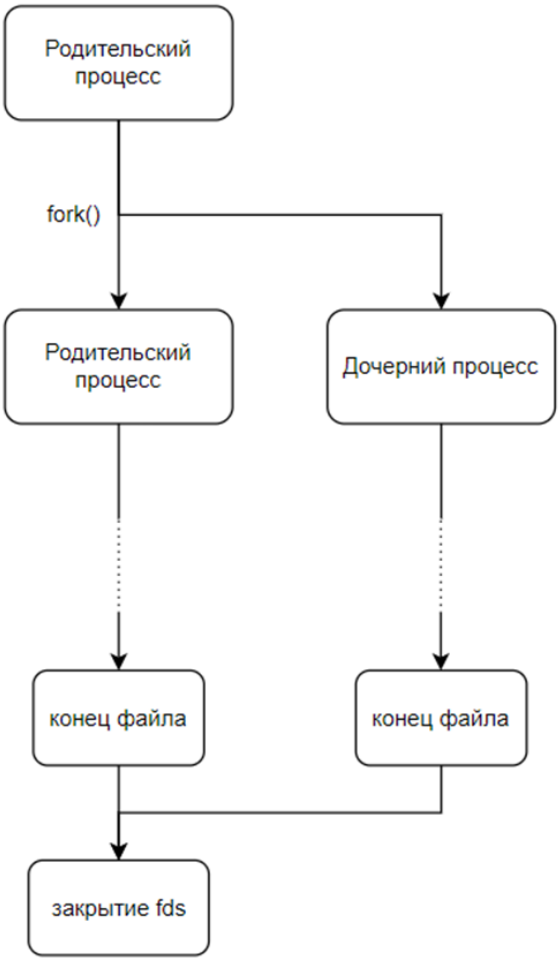

Рис. 1. Взаимодействие процессов

Описание основных используемых структур данных

Для хранения **pid** процессов используются переменные типа **pid_t**. **pid_t** - целое число со знаком, который способен представить ID процесса.

Блок-схема

На рисунках 2, 3, 4 представлена блок-схема программы.

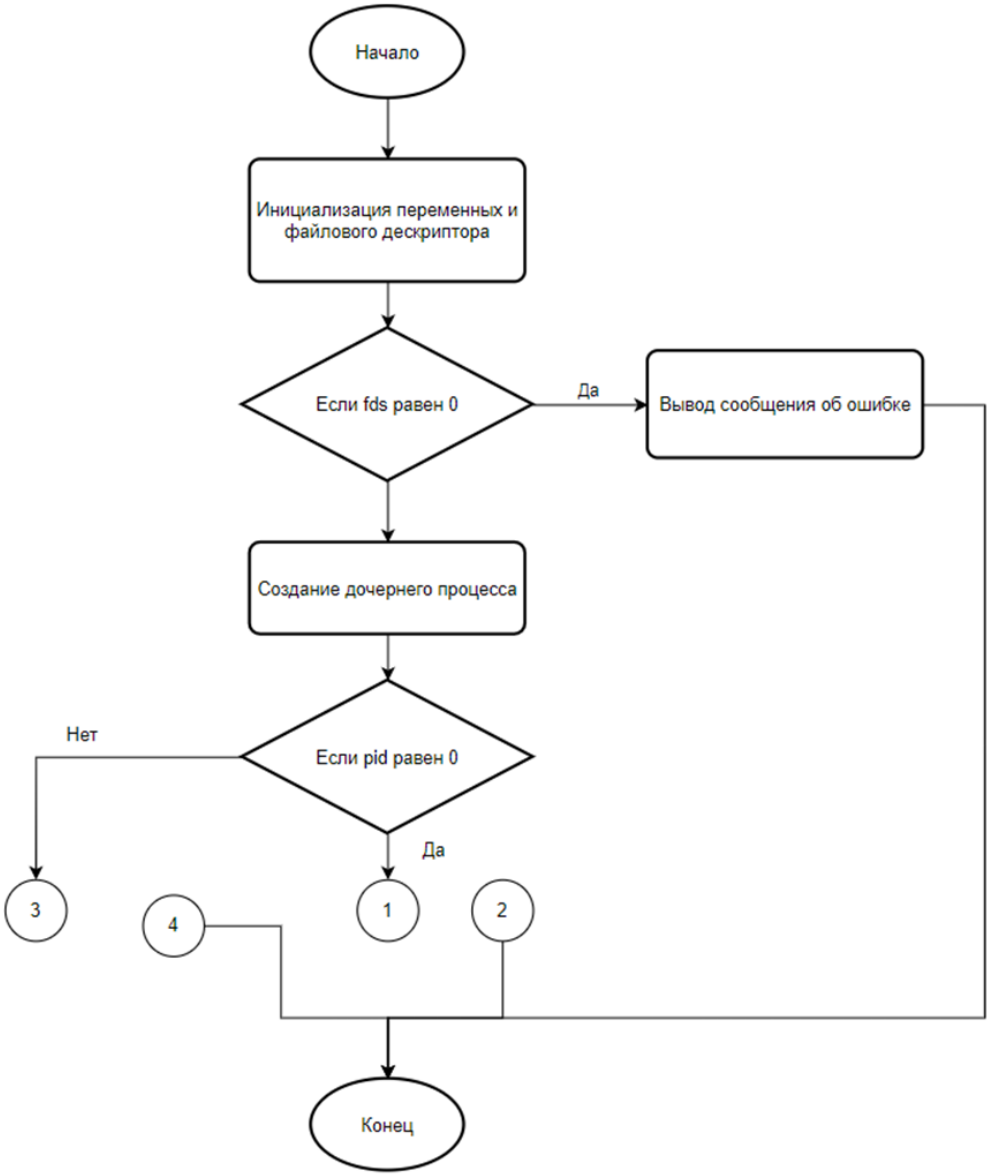

Рис. 2. Блок схема алгоритма работы программы

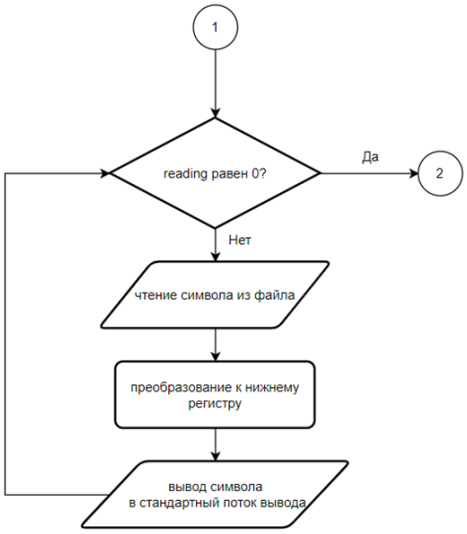

Рис. 3. Блок схема процесса-сына

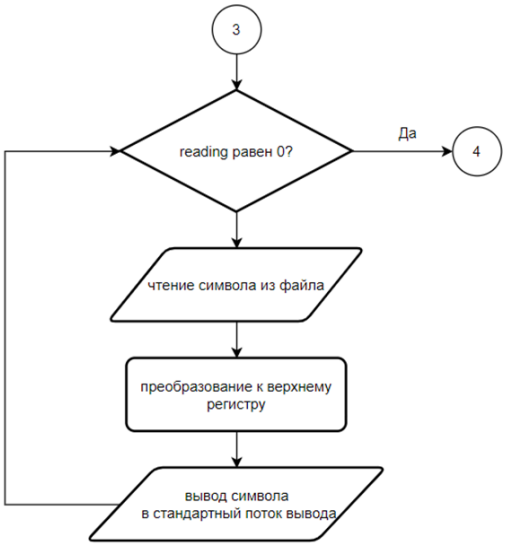

Рис. 4. Блок схема процесса-отца

Примеры работы программы

На рисунках 5 и 6 приведен пример работы программы, когда в нее поданы корректные аргументы, т.е. имя существующего файла.

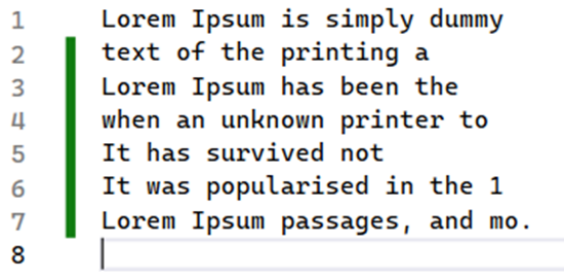

Рис. 5. Пример работы программы №1: входные данные

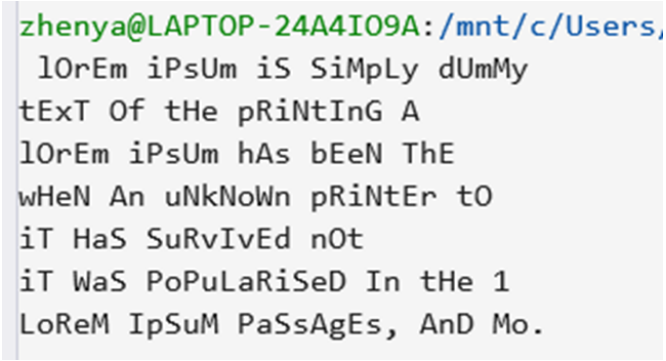

Рис. 6. Пример работы программы №1: выходные данные

На рисунке 7 приведены 2 примера работы программы, в которую подано неправильное количество аргументов или неправильное имя файла.

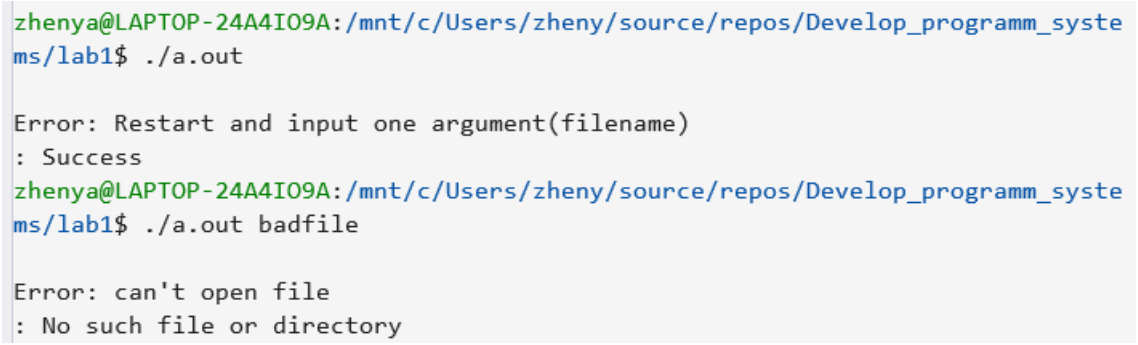

Рис. 7. Пример работы программы №2: при неправильном количестве аргументов и неверном имени входного файла

Текст программы

Ниже в листинге 1 представлен текст программы.

Листинг 1. Листинг программы

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#define sleep_delay 50000
#define _GNU_SOURCE

const char* OUT_ERR_1 = "\nError: Restart and input one argument(filename)\n";//error message
const char* OUT_ERR_2 = "\nError: can't open file\n";//error message
const char* OUT_ERR_3 = "\nError: can't read file\n";//error message
const char* OUT_ERR_4 = "\nError: can't output to the standard output stream\n";//error message
//function for writing lower symbols to out stream
int output_to_stream_lower(int fds, int ppid, int reading) {
  char symbol = ' ';
  while (reading) {
    //checking for an error when reading a file
    if ((reading = read(fds, &symbol, 1)) < 0) {
      perror(OUT_ERR_3);//error message output
      kill(ppid, SIGKILL);//in case of an error, the program terminates
      return -1;
    }
    //symbol -> lower symbol
    symbol = tolower(symbol);
    if (write(1, &symbol, 1) < 0) {//checking for an error when writing a file
      perror(OUT_ERR_4);//error message output
      kill(ppid, SIGKILL);//in case of an error, the program terminates
      return -1;
    }

    usleep(sleep_delay);

  }
  close(fds);
  exit(0);
```

```
39 }
40 //function for writing upper symbols to out stream
41 int output_to_stream_upper(int fds, int pid, int reading) {
42   char symbol = ' ';
43   while (reading) {
44     //checking for an error when reading a file
45     if ((reading = read(fds, &symbol, 1)) < 0) {
46       perror(OUT_ERR_3);//error message output
47       kill(pid, SIGKILL); //in case of an error, the program terminates
48       return -1;
49     }
50     //symbol -> upper symbol
51     symbol = toupper(symbol);
52     if (write(1, &symbol, 1) < 0) {//checking for an error when writing a file
53       perror(OUT_ERR_4);//error message output
54       kill(pid, SIGKILL);//in case of an error, the program terminates
55       return -1;
56     }
57     usleep(sleep_delay);
58   }
59   wait(0);
60   close(fds);
61 }
62 int main(int argc, char* argv[]) {
63
64
65   int fds = 0;
66   int reading = 1;
67
68   //checking for the number of arguments
69   if (argc < 2) {
70     perror(OUT_ERR_1);//error message output
71     return -1;
72   }
73   //checking for an error when open a file
74   if ((fds = open(argv[1], O_RDONLY)) == -1) {
75     perror(OUT_ERR_2);//error message output
76     return -1;
77   }
78
79   pid_t ppid = getpid(), pid;
80   pid = fork();
81   int i = 0;
82   if (pid == 0) {//child
83     if (output_to_stream_lower(fds, ppid, reading) == -1)return -1;
84   }
```

```
85   else {//parent
86     if (output_to_stream_upper(fds, pid, reading) == -1) return -1;
87   }
88 }
89 }
```