



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

---

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

## Отчёт

### по лабораторной работе №4

Название «Системные вызовы Unix: fork, exec»

---

Дисциплина «Операционные системы»

---

Студент ИУ7-55Б

---

\_\_\_\_\_  
(подпись, дата)

Клименко А.К.  
(Фамилия И.О.)

---

Преподаватель

\_\_\_\_\_  
(подпись, дата)

Рязанова Н.Ю.  
(Фамилия И.О.)

---

Москва, 2021

## Содержание

1	Теоритические сведения . . . . .	3
1.1	fork . . . . .	3
1.2	exec . . . . .	5
2	Задания . . . . .	7
2.1	Задание 1 . . . . .	7
2.2	Задание 2 . . . . .	9
2.3	Задание 3 . . . . .	11
2.4	Задание 4 . . . . .	13
2.5	Задание 5 . . . . .	15

# 1 Теоритические сведения

## 1.1 fork

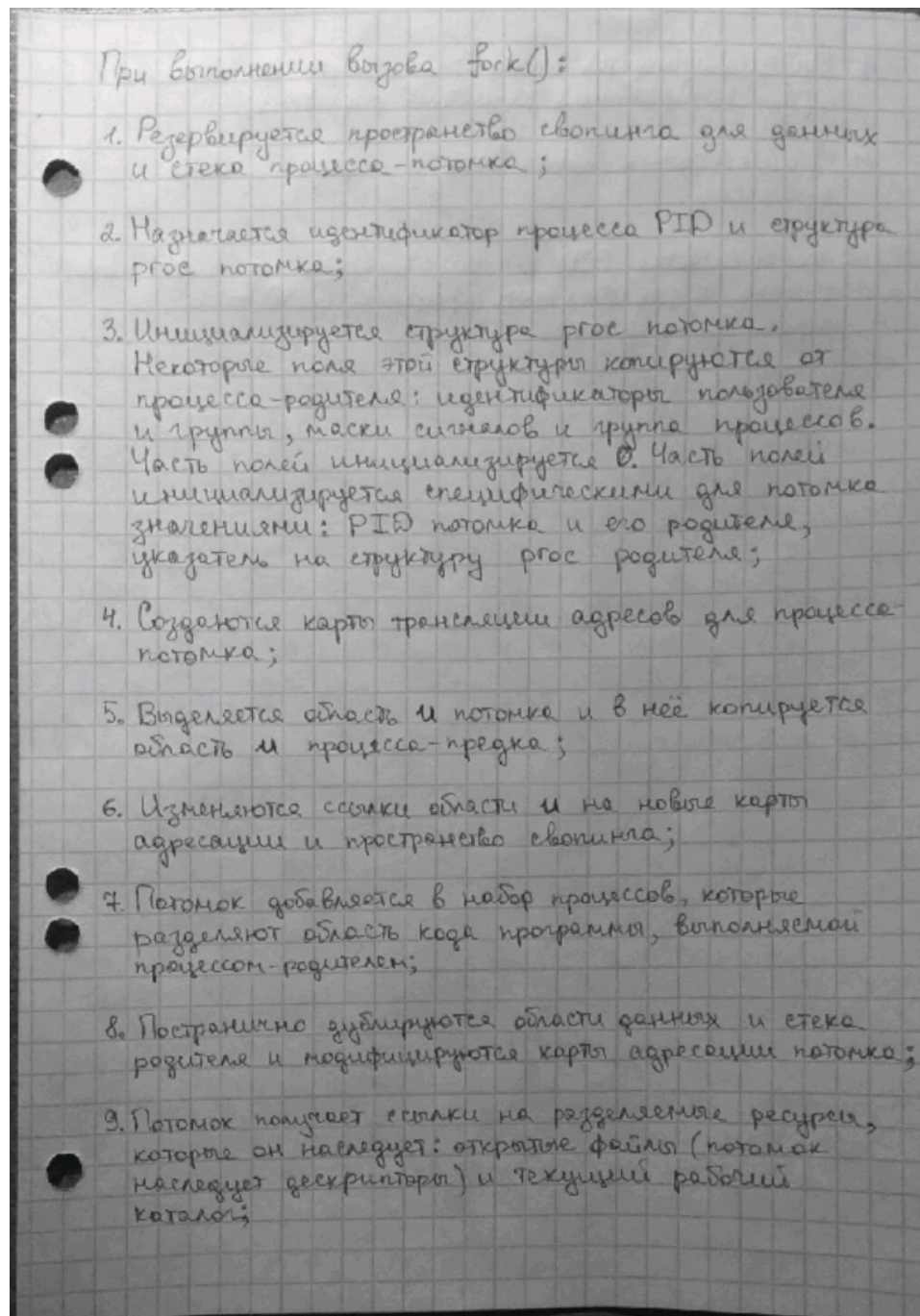


Рисунок 1.1 — Алгоритм работы системного вызова `fork` (Часть 1).

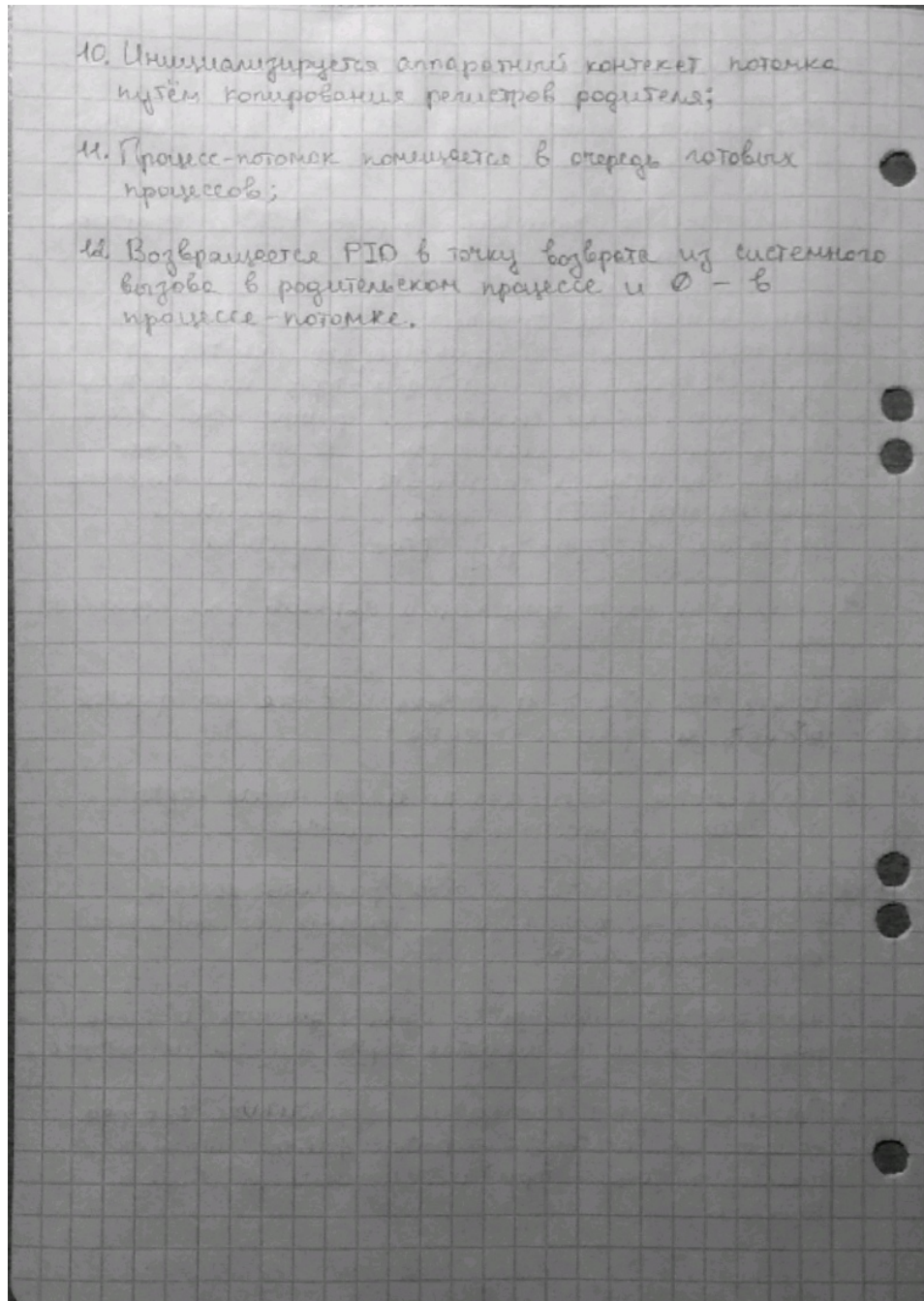


Рисунок 1.2 — Алгоритм работы системного вызова fork (Часть 2).

## 1.2 exes

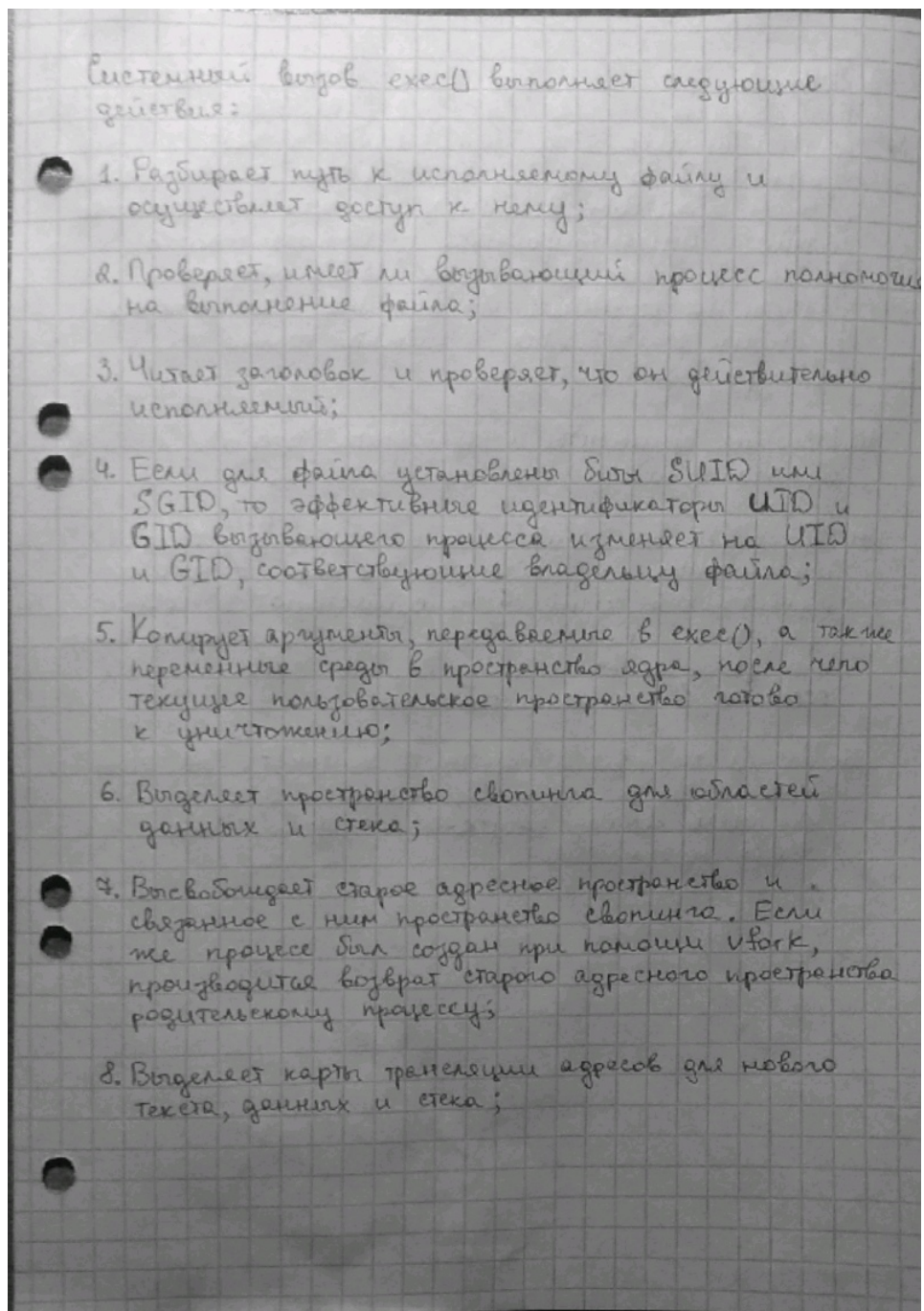


Рисунок 1.3 — Алгоритм работы системного вызова `exes` (Часть 1).



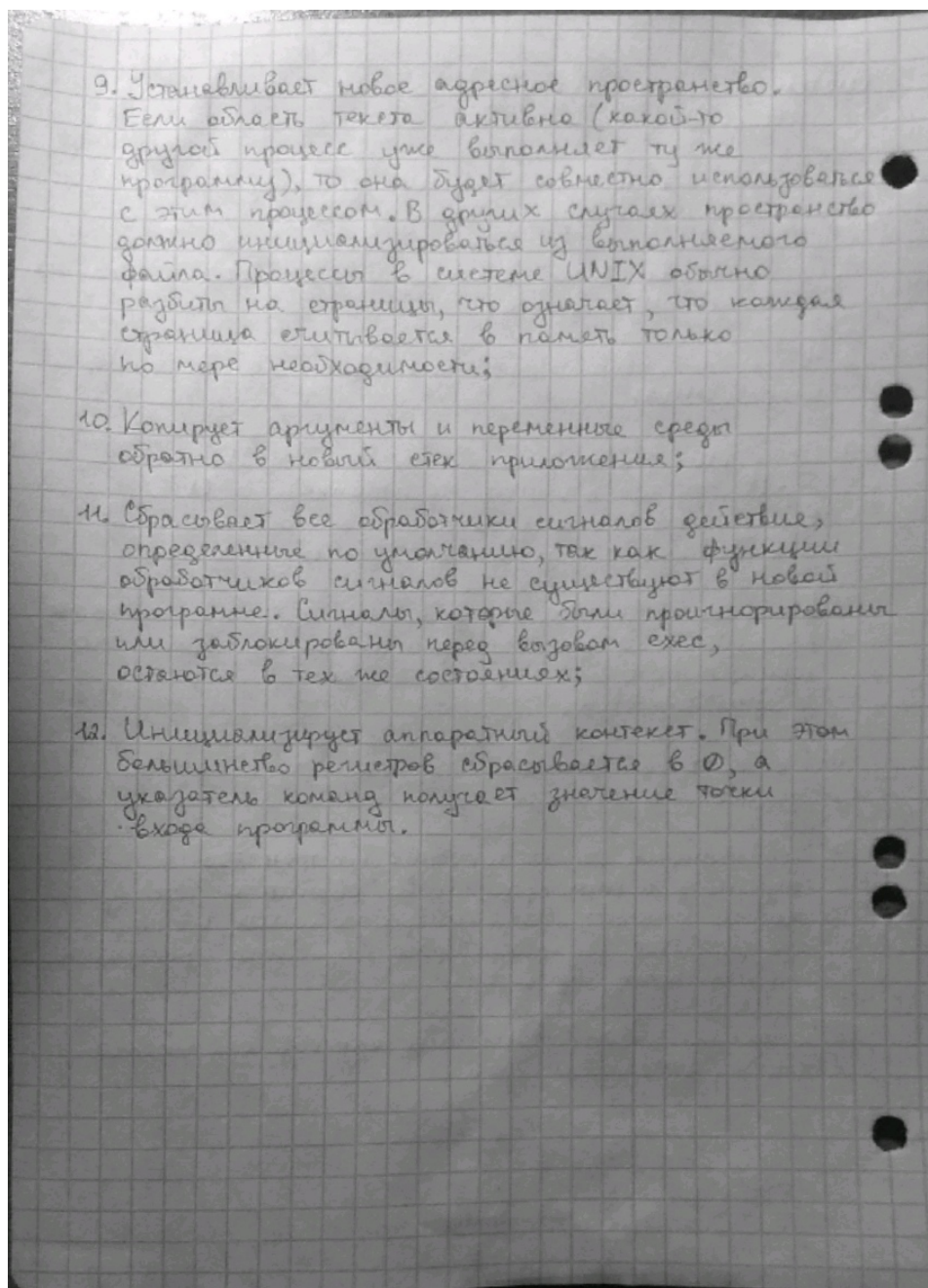


Рисунок 1.4 — Алгоритм работы системного вызова exec (Часть 2).

## 2 Задания

### 2.1 Задание 1

Написать программу, запускающую не менее двух новых процессов системным вызовом `fork()`. В предке вывести собственный идентификатор, идентификатор группы и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка и идентификатор группы. Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника. В потомках вызывается `sleep()`. Чтобы предок гарантированно завершился раньше своих потомков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе. Продемонстрировать «усыновление». Для этого надо в потомках вывести идентификаторы: собственный, предка, группы до блокировки и после блокировки.

Листинг 2.1 — Исходный код для задания 1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5
6  int main(void) {
7      printf("Предок:  pid=%d gid=%d\n", getpid(), getpgrp());
8
9      for (size_t i = 0; i < 5; i++) {
10         pid_t child_pid = fork();
11         if (child_pid == -1) {
12             printf("Не удалось выполнить fork().\n");
13             exit(EXIT_FAILURE);
14         } else if (child_pid == 0) {
15             printf("Потомок: pid=%d ppid=%d gid=%d\n", getpid(), getppid(), getpgrp());
16             sleep(1);
17             printf("pid=%d ppid=%d gid=%d\n", getpid(), getppid(), getpgrp());
18             exit(EXIT_SUCCESS);
19         }
20     }
21
22     return EXIT_SUCCESS;
23 }
```

```
rulldeef@pop-os:~/Projects/os-lab/lab_04$ gcc -o task1 task1.c
rulldeef@pop-os:~/Projects/os-lab/lab_04$ ./task1
Предок: pid=127576 gid=127576
Потомок: pid=127577 ppid=127576 gid=127576
Потомок: pid=127578 ppid=127576 gid=127576
Потомок: pid=127579 ppid=127576 gid=127576
Потомок: pid=127580 ppid=127576 gid=127576
Потомок: pid=127581 ppid=127576 gid=127576
rulldeef@pop-os:~/Projects/os-lab/lab_04$ pid=127578 ppid=2977 gid=127576
pid=127579 ppid=2977 gid=127576
pid=127577 ppid=2977 gid=127576
pid=127581 ppid=2977 gid=127576
pid=127580 ppid=2977 gid=127576
```

Рисунок 2.1 — Скриншот, демонстрирующий работу программы.



## 2.2 Задание 2

написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов `wait()`. Убедиться, что в этом случае идентификатор процесса потомка на 1 больше идентификатора процесса-предка.

Предок ждет завершения своих потомком, используя системный вызов `wait()`. Вывод соответствующих сообщений на экран. В программе необходимо, чтобы предок выполнял анализ кодов завершения потомков.

Листинг 2.2 — Исходный код для задания 2.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  int main(void) {
8      printf("Предок: pid=%d gid=%d\n", getpid(), getpgrp());
9
10     for (size_t i = 0; i < 5; i++) {
11         pid_t child_pid = fork();
12         if (child_pid == -1) {
13             printf("Не удалось выполнить fork().\n");
14             exit(EXIT_FAILURE);
15         } else if (child_pid == 0) {
16             printf("Потомок #%ld: pid=%d ppid=%d gid=%d\n", i, getpid(), getppid(), getpgrp());
17             exit(100 + i);
18         }
19     }
20
21     for (size_t i = 0; i < 5; i++) {
22         int status;
23         pid_t child_pid = wait(&status);
24         printf("Потомок с pid=%d ", child_pid);
25
26         if (WIFEXITED(status))
27             printf("завершился с кодом %d.\n", WEXITSTATUS(status));
28         else if (WIFSIGNALED(status))
29             printf("остановлен сигналом с кодом %d\n", WTERMSIG(status));
30         else if (WIFSTOPPED(status))
31             printf("остановлен с кодом %d\n", WSTOPSIG(status));
32     }
33
34     return EXIT_SUCCESS;
35 }
```

```
rulldееf@pop-os:~/Projects/os-lab/lab_04$ gcc -o task2 task2.c
rulldееf@pop-os:~/Projects/os-lab/lab_04$ ./task2
Предок: pid=129341 gid=129341
Потомок №0: pid=129342 ppid=129341 gid=129341
Потомок №1: pid=129343 ppid=129341 gid=129341
Потомок №2: pid=129344 ppid=129341 gid=129341
Потомок с pid=129342 завершился с кодом 100.
Потомок №3: pid=129345 ppid=129341 gid=129341
Потомок с pid=129343 завершился с кодом 101.
Потомок №4: pid=129346 ppid=129341 gid=129341
Потомок с pid=129344 завершился с кодом 102.
Потомок с pid=129345 завершился с кодом 103.
Потомок с pid=129346 завершился с кодом 104.
rulldееf@pop-os:~/Projects/os-lab/lab_04$
```

Рисунок 2.2 — Скриншот, демонстрирующий работу программы.

## 2.3 Задание 3

Написать программу, в которой процесс-потомок вызывает системный вызов `exec()`, а процесс-предок ждет завершения процесса-потомка. Следует создать не менее двух потомков.

Потомки переходят на выполнение других программ, которые передаются системному вызову `exec()` в качестве параметра. Потомки должны выполнять разные программы. Предок ждет завершения своих потомков с анализом кодов завершения. На экран выводятся соответствующие сообщения.

Листинг 2.3 — Исходный код для задания 3.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  void execute_program(const char* filepath, const char* filename, const char* arg);
8
9  int main(void) {
10     execute_program("./lister", "lister", "list_in.txt");
11     execute_program("./pysort.py", "pysort.py", "list_in.txt");
12     return EXIT_SUCCESS;
13 }
14
15 void execute_program(const char* filepath, const char* filename, const char* arg) {
16     pid_t child_pid = fork();
17
18     if (child_pid == -1) {
19         printf("Не удалось выполнить fork().\n");
20         exit(EXIT_FAILURE);
21     } else if (child_pid == 0)
22         execl(filepath, filename, arg, NULL);
23
24     int status;
25     waitpid(child_pid, &status, 0);
26
27     printf("Потомок с pid=%d ", child_pid);
28     if (WIFEXITED(status))
29         printf("завершился с кодом %d.\n", WEXITSTATUS(status));
30     else if (WIFSIGNALED(status))
31         printf("остановлен сигналом с кодом %d\n", WTERMSIG(status));
32     else if (WIFSTOPPED(status))
33         printf("остановлен с кодом %d\n", WSTOPSIG(status));
34 }
```

```
rulldееf@pop-os:~/Projects/os-lab/lab_04$ gcc -o task3 ./task3.c
rulldееf@pop-os:~/Projects/os-lab/lab_04$ cat list_in.txt
8 2 4 -2 11 0 3 -7 1
rulldееf@pop-os:~/Projects/os-lab/lab_04$ ./task3
1 -7 3 0 11 -2 4 2 8
Потомок с pid=136068 завершился с кодом 0.
-7 -2 0 1 2 3 4 8 11
Потомок с pid=136069 завершился с кодом 0.
rulldееf@pop-os:~/Projects/os-lab/lab_04$
```

Рисунок 2.3 — Скриншот, демонстрирующий работу программы.

## 2.4 Задание 4

Написать программу, в которой предок и потомок обмениваются сообщением через программный канал. Причем оба потомка пишут свои сообщения в один программный канал, а предок их считывает из канала. Потомки должны посылать предку разные сообщения по содержанию и размеру. Предок считывает сообщения от потомков и выводит их на экран. Предок ждет завершения своих потомков и анализирует код их завершения. Вывод соответствующих сообщений на экран.

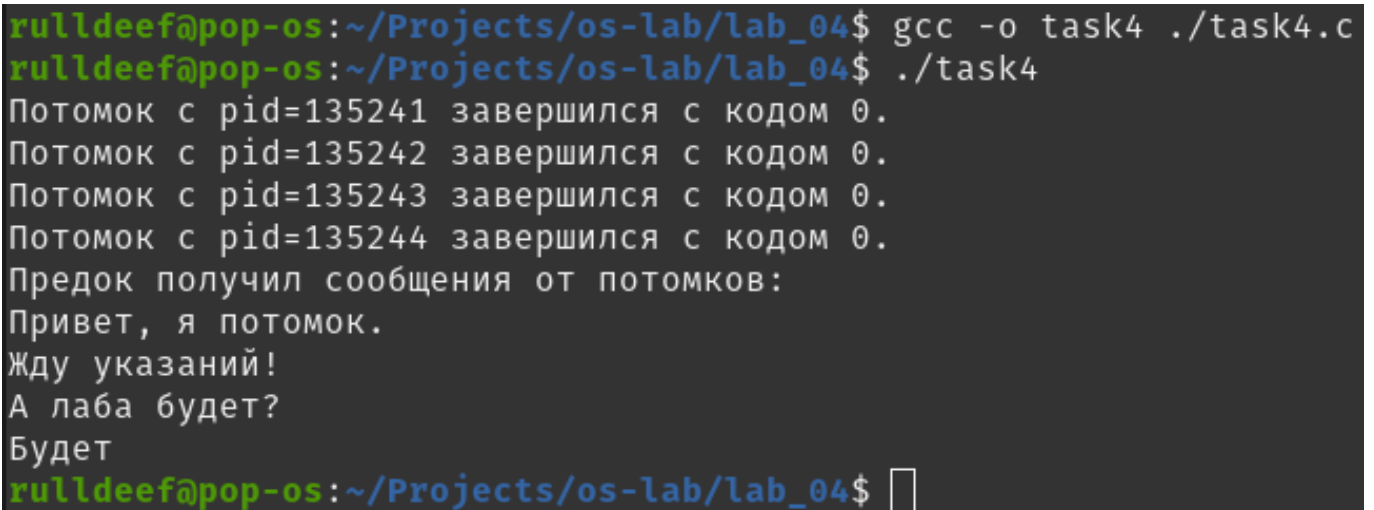
Листинг 2.4 — Исходный код для задания 4 (Часть 1).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7
8  int main(void) {
9      int pipefd[2];
10     if (pipe(pipefd) == -1) {
11         printf("Не удалось создать программный канал.\n");
12         exit(EXIT_FAILURE);
13     }
14
15     const char* messages[] = {
16         "Привет, я потомок.\n",
17         "Жду указаний!\n",
18         "А лаба будет?\n",
19         "Будет\n"
20     };
21     size_t n = sizeof(messages) / sizeof(messages[0]);
22     for (size_t i = 0; i < n; i++) {
23         pid_t child_pid = fork();
24         if (child_pid == -1) {
25             printf("Не удалось выполнить fork().\n");
26             exit(EXIT_FAILURE);
27         } else if (child_pid == 0) {
28             close(pipefd[0]); // закрытие канала для чтения
29             write(pipefd[1], messages[i], strlen(messages[i]));
30             exit(EXIT_SUCCESS);
31         }
32     }
```



Листинг 2.5 — Исходный код для задания 4 (Часть 2).

```
33     for (size_t i = 0; i < n; i++) {
34         int status;
35         pid_t child_pid = wait(&status);
36         printf("Потомок с pid=%d ", child_pid);
37
38         if (WIFEXITED(status))
39             printf("завершился с кодом %d.\n", WEXITSTATUS(status));
40         else if (WIFSIGNALED(status))
41             printf("остановлен сигналом с кодом %d\n", WTERMSIG(status));
42         else if (WIFSTOPPED(status))
43             printf("остановлен с кодом %d\n", WSTOPSIG(status));
44     }
45
46     close(pipefd[1]); // закрытие канала на запись
47
48     char buffer[256];
49     read(pipefd[0], buffer, 256);
50     printf("Предок получил сообщения от потомков:\n%s", buffer);
51
52     return EXIT_SUCCESS;
53 }
```



```
rulldeef@pop-os:~/Projects/os-lab/lab_04$ gcc -o task4 ./task4.c
rulldeef@pop-os:~/Projects/os-lab/lab_04$ ./task4
Потомок с pid=135241 завершился с кодом 0.
Потомок с pid=135242 завершился с кодом 0.
Потомок с pid=135243 завершился с кодом 0.
Потомок с pid=135244 завершился с кодом 0.
Предок получил сообщения от потомков:
Привет, я потомок.
Жду указаний!
А лаба будет?
Будет
rulldeef@pop-os:~/Projects/os-lab/lab_04$
```

Рисунок 2.4 — Скриншот, демонстрирующий работу программы.

## 2.5 Задание 5

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

Предок и потомки аналогично заданию 4 обмениваются сообщениями через неименованный программный канал. В программу включается собственный обработчик сигнала. С помощью сигнала меняется ход выполнения программы. При получении сигнала потомки записывают сообщения в канал, если сигнал не поступает, то не записывают. Предок ждет завершения своих потомков и анализирует коды их завершений. Вывод соответствующих сообщений на экран.

Листинг 2.6 — Исходный код для задания 5 (Часть 1).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7  #include <sys/signal.h>
8
9  int write_enabled = 0;
10
11 void child_sig_handler(int sig) {
12     write_enabled = 1;
13 }
14
15 int main(void) {
16     int pipefd[2];
17     if (pipe(pipefd) == -1) {
18         printf("Не удалось создать программный канал.\n");
19         exit(EXIT_FAILURE);
20     }
21
22     const char* messages[] = {
23         "Привет, я потомок.\n",
24         "Жду указаний!\n",
25         "А лаба будет?\n",
26         "Будет\n"
27     };
28     size_t n = sizeof(messages) / sizeof(messages[0]);
29     for (size_t i = 0; i < n; i++) {
30         pid_t child_pid = fork();
31         if (child_pid == -1) {
32             printf("Не удалось выполнить fork().\n");
33             exit(EXIT_FAILURE);
```

Листинг 2.7 — Исходный код для задания 5 (Часть 2).

```

34     } else if (child_pid == 0) {
35         close(pipefd[0]); // закрытие канала для чтения
36         // назначение обработчика сигнала для процесса-потомка
37         signal(SIGINT, child_sig_handler);
38         sleep(i);
39         if (write_enabled)
40             write(pipefd[1], messages[i], strlen(messages[i]));
41         exit(EXIT_SUCCESS);
42     }
43 }
44
45 close(pipefd[1]); // закрытие канала для записи
46 // игнорирование сигнала SIGINT в родительском процессе
47 signal(SIGINT, SIG_IGN);
48
49 for (size_t i = 0; i < n; i++) {
50     int status;
51     pid_t child_pid = wait(&status);
52     printf("Потомок с pid=%d ", child_pid);
53
54     if (WIFEXITED(status))
55         printf("завершился с кодом %d.\n", WEXITSTATUS(status));
56     else if (WIFSIGNALED(status))
57         printf("остановлен сигналом с кодом %d\n", WTERMSIG(status));
58     else if (WIFSTOPPED(status))
59         printf("остановлен с кодом %d\n", WSTOPSIG(status));
60 }
61
62 char buffer[256] = {'\0'};
63 if (read(pipefd[0], buffer, 256))
64     printf("Предок получил сообщения от потомков:\n%s", buffer);
65 else
66     printf("Предок не получил сообщений от потомков.\n");
67
68 return EXIT_SUCCESS;
69 }

```

```
rullddeef@pop-os:~/Projects/os-lab/lab_04$ gcc -o task5 task5.c
rullddeef@pop-os:~/Projects/os-lab/lab_04$ ./task5
Потомок с pid=128901 завершился с кодом 0.
Потомок с pid=128902 завершился с кодом 0.
^СПотомок с pid=128903 завершился с кодом 0.
Потомок с pid=128904 завершился с кодом 0.
Предок получил сообщения от потомков:
Будет
А лаба будет?
rullddeef@pop-os:~/Projects/os-lab/lab_04$
```

Рисунок 2.5 — Скриншот, демонстрирующий работу программы. В процессе работы программы был послан сигнал SIGINT.

```
rullddeef@pop-os:~/Projects/os-lab/lab_04$ gcc -o task5 task5.c
rullddeef@pop-os:~/Projects/os-lab/lab_04$ ./task5
Потомок с pid=128804 завершился с кодом 0.
Потомок с pid=128805 завершился с кодом 0.
Потомок с pid=128806 завершился с кодом 0.
Потомок с pid=128807 завершился с кодом 0.
Предок не получил сообщений от потомков.
rullddeef@pop-os:~/Projects/os-lab/lab_04$
```

Рисунок 2.6 — Скриншот, демонстрирующий работу программы. В процессе работы программы сигналы не посылались.