**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»

# Отчет по лабораторной работе №4 по дисциплине "Операционные системы"

**Тема** Процессы. Системные вызовы fork() и exec()

**Студент** Артемьев И. О.

**Группа** ИУ7-53Б

**Оценка (баллы)** _____

**Преподаватели** Рязанова Н. Ю.

Москва — 2021 г.

# Задание №1

Процессы-сироты. В программе создаются не менее двух потомков. В потомках вызывается sleep(). Чтобы предок гарантированно завершился раньше своих помков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе.

Листинг 1: Процессы-сироты

```c
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    int childpid_first = fork(), childpid_second;

    if (childpid_first == -1)
    {
        perror("Can't fork\n");
        return -1;
    }
    else if (childpid_first == 0)
    {
        sleep(1);
        printf("\nFirst child: id: %d ppid: %d  pgroup: %d\n", getpid(),
           getppid(), getpgrp());
        return 0;
    }

    childpid_second = fork();
    if (childpid_second == -1)
    {
        perror("Can't fork\n");
        return -1;
    }
    else if (childpid_second == 0)
    {
        sleep(1);
        printf("\nSecond child: id: %d ppid: %d  pgroup: %d\n\n", getpid(),
           getppid(), getpgrp());
        return 0;
    }

    printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n",
       getpid(), getpgrp(), childpid_first, childpid_second);

    return 0;
}
```

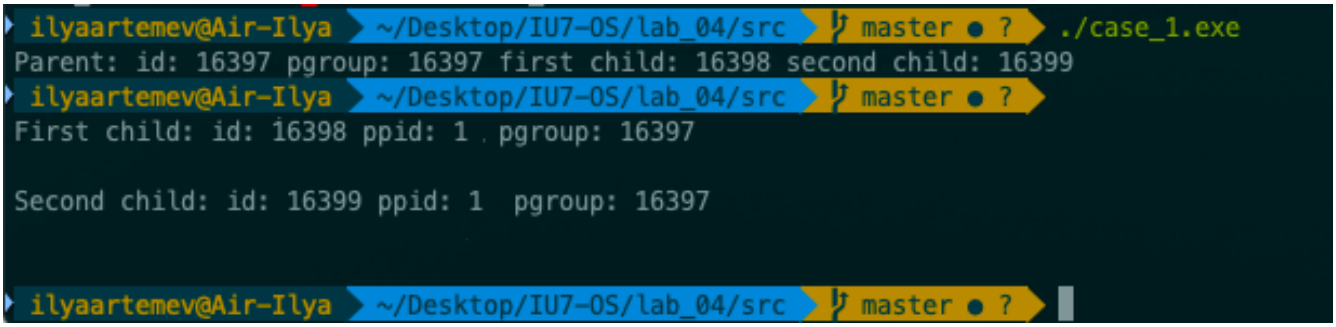


Рис. 1: Демонстрация работы программы (задание №1).

# Задание №2

Предок ждет завершения своих потомком, используя системный вызов wait(). Вывод соответствующих сообщений на экран.

Листинг 2: Вызов функции wait()

```c
#include "status.h"

int main(void)
{
    int status;
    int childpid_first = fork(), childpid_second;

    if (childpid_first == -1)
    {
        perror("Can't fork\n");
        return -1;
    }
    else if (childpid_first == 0)
    {

        printf("\nFirst child: id: %d ppid: %d  pgroup: %d\n", getpid(),
           getppid(), getpgrp());
        return 0;
    }

    childpid_second = fork();
    if (childpid_second == -1)
    {
        perror("Can't fork\n");
        return -1;
    }
    else if (childpid_second == 0)
    {

        printf("\nSecond child: id: %d ppid: %d  pgroup: %d\n\n", getpid(),
           getppid(), getpgrp());
        return 0;
    }

    wait(&status);
    check_status(status);
    wait(&status);
    check_status(status);

    printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n",
       getpid(), getpgrp(), childpid_first, childpid_second);

    return 0;
}
```

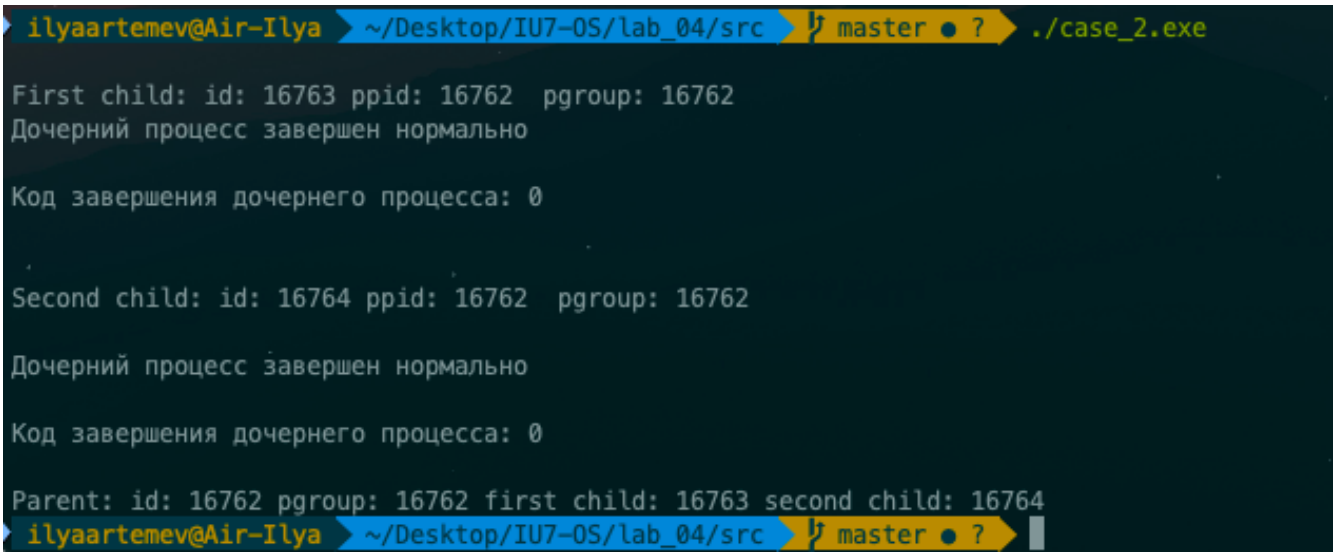


Рис. 2: Демонстрация работы программы (задание №2).

# Задание №3

Потомки переходят на выполнение других программ. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 3: Вызов функции execlp()

```
#include "status.h"

int main(void)
{
    int status;
    int childpid_first = fork(), childpid_second;

    if (childpid_first == -1)
    {
        perror("Can't fork\n");
        return -1;
    }
    else if (childpid_first == 0)
    {
        printf("\nFirst child: id: %d ppid: %d  pgroup: %d\n", getpid(),
           getppid(), getpgrp());

        if (execlp("./levenstein.exe", "./levenstein.exe", NULL) == -1)
        {
            perror("Can't execlp first child\n\n");
            return -1;
        }

        return 0;
    }

    childpid_second = fork();
    if (childpid_second == -1)
    {
        perror("Can't fork\n");
        return -1;
    }
    else if (childpid_second == 0)
    {
        printf("\nSecond child: id: %d ppid: %d  pgroup: %d\n\n", getpid(),
           getppid(), getpgrp());

        sleep(8);
        if (execlp("./sort.exe", "./sort.exe", NULL) == -1)
        {
            perror("Can't execlp second child\n\n");
            return -1;
        }

```

```
43          return 0;
44      }
45
46      wait(&status);
47      check_status(status);
48      wait(&status);
49      check_status(status);
50
51      printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n",
            getpid(), getpgrp(), childpid_first, childpid_second);
52
53      return 0;
54  }
```

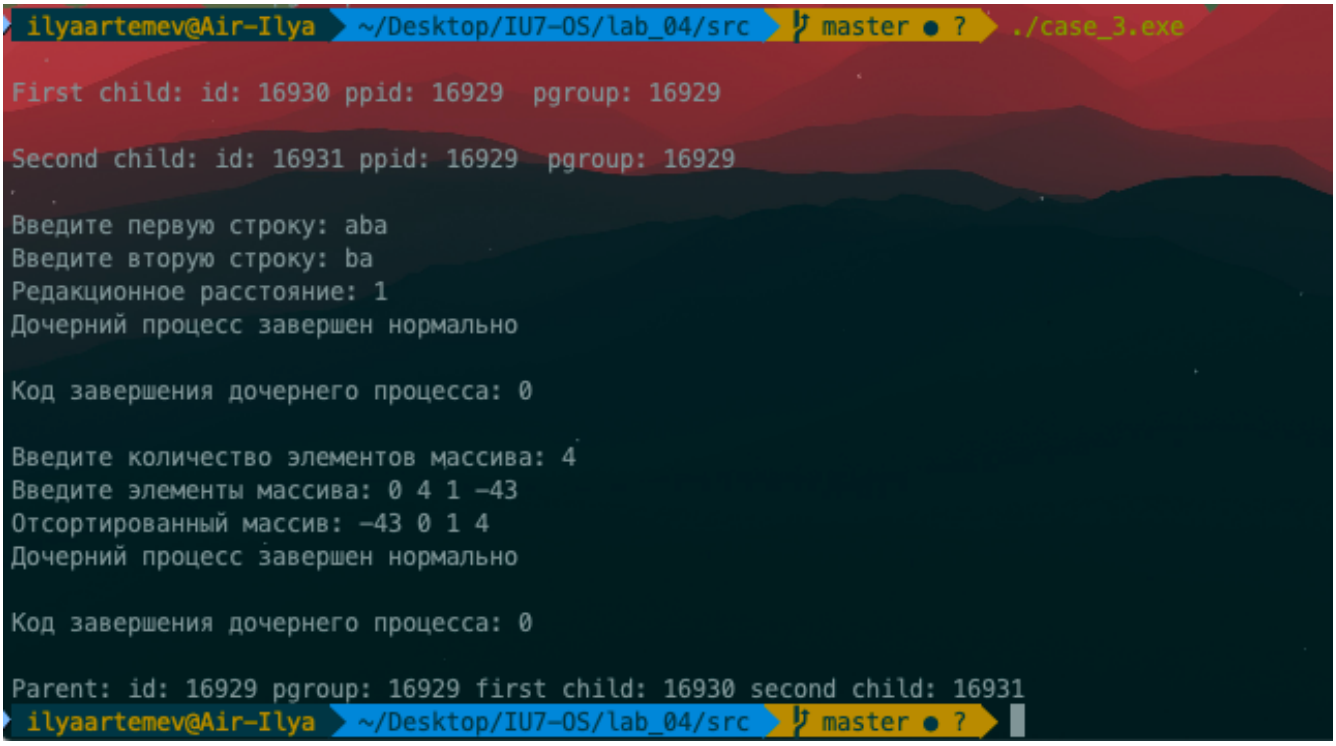


Рис. 3: Демонстрация работы программы (задание №3).

# Задание №4

Предок и потомки обмениваются сообщениями через неименованный программный канал. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 4: Использование pipe

```
#include "status.h"

int main(void)
{
    const char *msg_1 = "Overleaf is used by over eight million students and
        academics at 3,600 institutions worldwide.\n";
    const char *msg_2 = "Just want to say that I am really grateful for
       Overleaf, it has enabled a slew of research and teaching development
       in my work that would have been annoyingly difficult before. Even
       people who dont know LaTeX are participating with me on research
       proposals and thats saying something if you know what LaTeX is
       like for the uninitiated.\n";

    char readbuffer[1000];
    int fd[2];

    int status;
    int childpid_first, childpid_second;

    if (pipe(fd) == -1)
    {
        printf("1\n");
        perror("Can't pipe\n");
        return -1;
    }

    childpid_first = fork();
    if (childpid_first == -1)
    {
        perror("Cant fork.\n");
        return -1;
    }
    else if (childpid_first == 0)
    {
        close(fd[0]);
        write(fd[1], msg_1, strlen(msg_1));

        return 0;
    }

    childpid_second = fork();
    if (childpid_second == -1)
    {
        perror("Can't fork\n");
```

```
39          return -1;
40      }
41      else if (childpid_second == 0)
42      {
43          close(fd[0]);
44          write(fd[1], msg_2, strlen(msg_2));
45
46          return 0;
47      }
48
49      wait(&status);
50      check_status(status);
51      wait(&status);
52      check_status(status);
53
54      printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n\n",
            getpid(), getpgrp(), childpid_first, childpid_second);
55
56      close(fd[1]);
57      read(fd[0], readbuffer, sizeof(readbuffer));
58      printf("%s", readbuffer);
59
60      return 0;
61  }
```

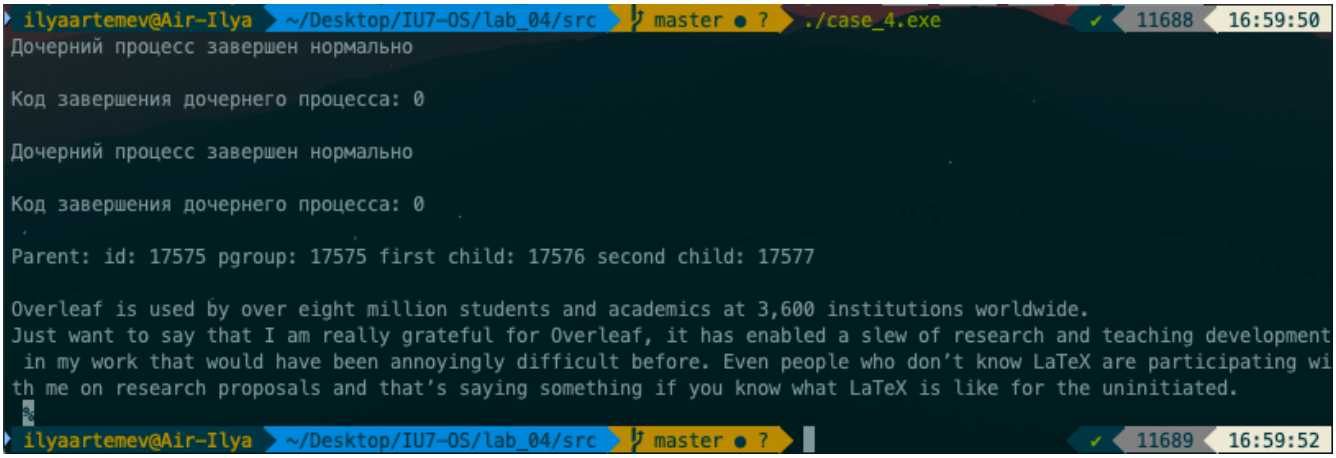


Рис. 4: Демонстрация работы программы (задание №4).

# Задание №5

Предок и потомки обмениваются сообщениями через неименованный программный канал. С помощью сигнала меняется ход выполнения программы. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 5: Использование сигналов

```
#include "status.h"

int flag = 0;

void ctrl_c(const int signal_numb)
{
    flag = 1;
    printf("\nSignal number: %d\n", signal_numb);
}

int main(void)
{
    const char *msg_1 = "Overleaf is used by over eight million students and
        academics at 3,600 institutions worldwide.\n";
    const char *msg_2 = "Just want to say that I am really grateful for
       Overleaf, it has enabled a slew of research and teaching development
       in my work that would have been annoyingly difficult before. Even
       people who dont know LaTeX are participating with me on research
       proposals and thats saying something if you know what LaTeX is
       like for the uninitiated.\n";

    char readbuffer[1000];
    int fd[2];

    int status;
    int childpid_first, childpid_second;

    signal(SIGINT, ctrl_c);

    if (pipe(fd) == -1)
    {
        perror("Can't pipe\n");
        return -1;
    }

    printf("If you want to write messages to the channel, press: \"CTRL+C
       \".\n");
    sleep(20);

    childpid_first = fork();
    if (childpid_first == -1)
    {
        perror("Cant fork.\n");
```

```
37          return -1;
38      }
39      else if (childpid_first == 0)
40      {
41          if (flag == 1)
42          {
43              close(fd[0]);
44              write(fd[1], msg_1, strlen(msg_1));
45          }
46
47          return 0;
48      }
49
50      childpid_second = fork();
51      if (childpid_second == -1)
52      {
53          perror("Can't fork\n");
54          return -1;
55      }
56      else if (childpid_second == 0)
57      {
58          if (flag == 1)
59          {
60              close(fd[0]);
61              write(fd[1], msg_2, strlen(msg_2));
62          }
63
64          return 0;
65      }
66
67      wait(&status);
68      check_status(status);
69      wait(&status);
70      check_status(status);
71
72      printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n\n",
            getpid(), getpgrp(), childpid_first, childpid_second);
73
74      if (flag == 1)
75      {
76          close(fd[1]);
77          read(fd[0], readbuffer, sizeof(readbuffer));
78          printf("%s", readbuffer);
79      }
80
81      return 0;
82  }
```

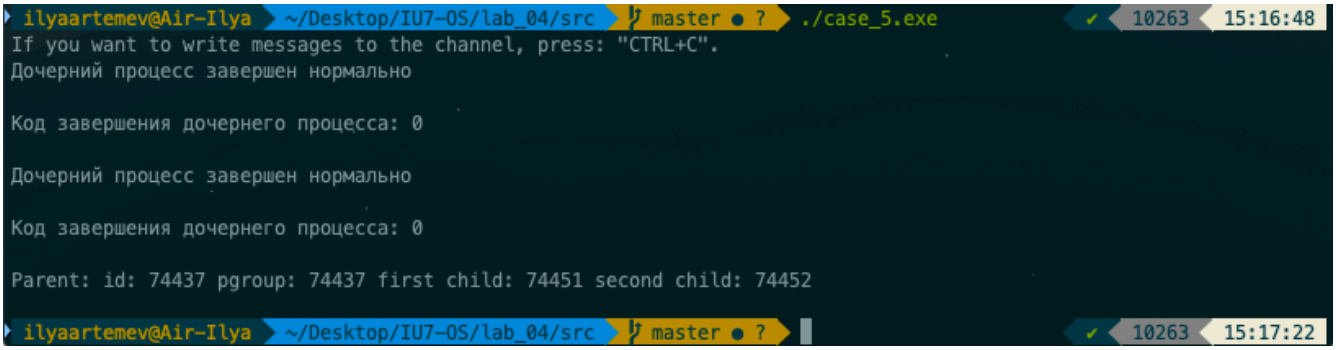


Рис. 5: Демонстрация работы программы, сигнал не вызывается (задание №5).


Рис. 6: Демонстрация работы программы, сигнал вызывается (задание №5).

# Дополнительные файлы

Листинг 6: Проверка статуса завершившегося дочернего процесса

```c
#include "status.h"

void check_status(const int status)
{
    if (WIFEXITED(status))
    {
        printf("Child process completed successfully\n\n");
        printf("Return code: %d\n\n", WEXITSTATUS(status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("The child process terminates with an un-intercepted signal\n
           \n");
        printf("Signal number: %d\n\n", WTERMSIG(status));
    }
    else if (WIFSTOPPED(status))
    {
        printf("The child process has stopped\n\n");
        printf("Signal number: %d\n\n", WSTOPSIG(status));
    }

    return;
}
```

Листинг 7: Хедер для подключения к файлам вызывающим check_status

```c
#ifndef __STATUS_H__
#define __STATUS_H__

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

void check_status(const int status);

#endif // __STATUS_H__
```

# Дополнительное задание

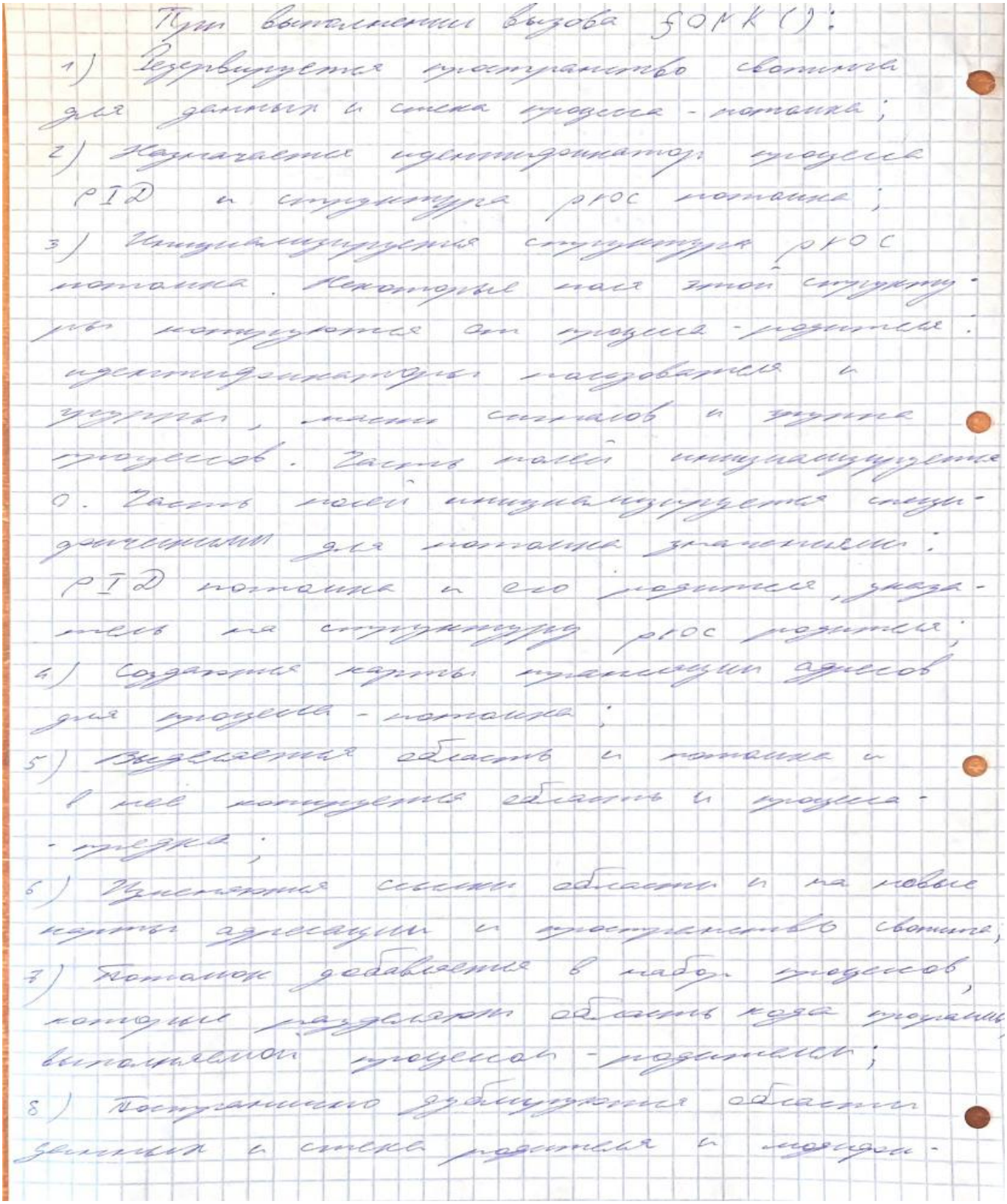


Рис. 7: Очередность при вызове fork().

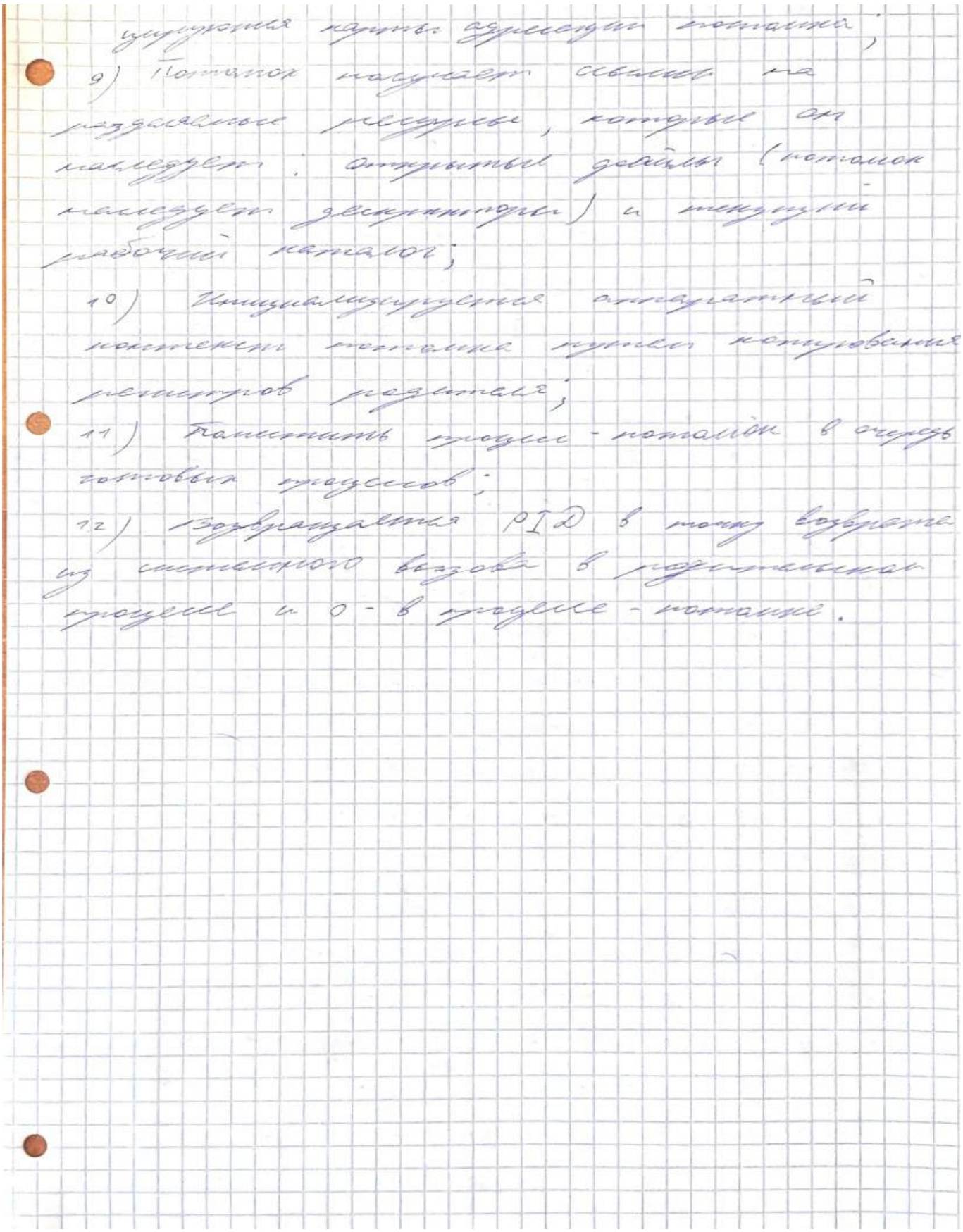


Рис. 8: Очередность при вызове fork().

При выполнении вызова exec():

1) Разбирает путь к исполняемому файлу и осуществляет доступ к нему.

2) Проверяет, имеет ли вызывающий процесс полномочия на выполнение файла.

3) Читает заголовок и проверяет, что он действительно исполняемый.

4) Если для файла установлены биты SUID или SGID, то эффективные идентификаторы UID и GID вызывающего процесса изменяют на UID и GID, соответствующие владельцу файла.

5) Копирует аргументы, переданные в exec, а также переменные среды в пространство ядра, после чего текущее адресное пространство полностью готово к уничтожению.

6) Освобождает пространство свопинга для области данных и стека.

7) Освобождает старые адресные пространства и связанные с ним пространства свопинга. Если не нашел был создан при вызове vfork, процесс возвращает старое адресное пространство родительскому процессу.

8) Выделяет карты страниц для новых адресов

Рис. 9: Очередность при вызове exec().

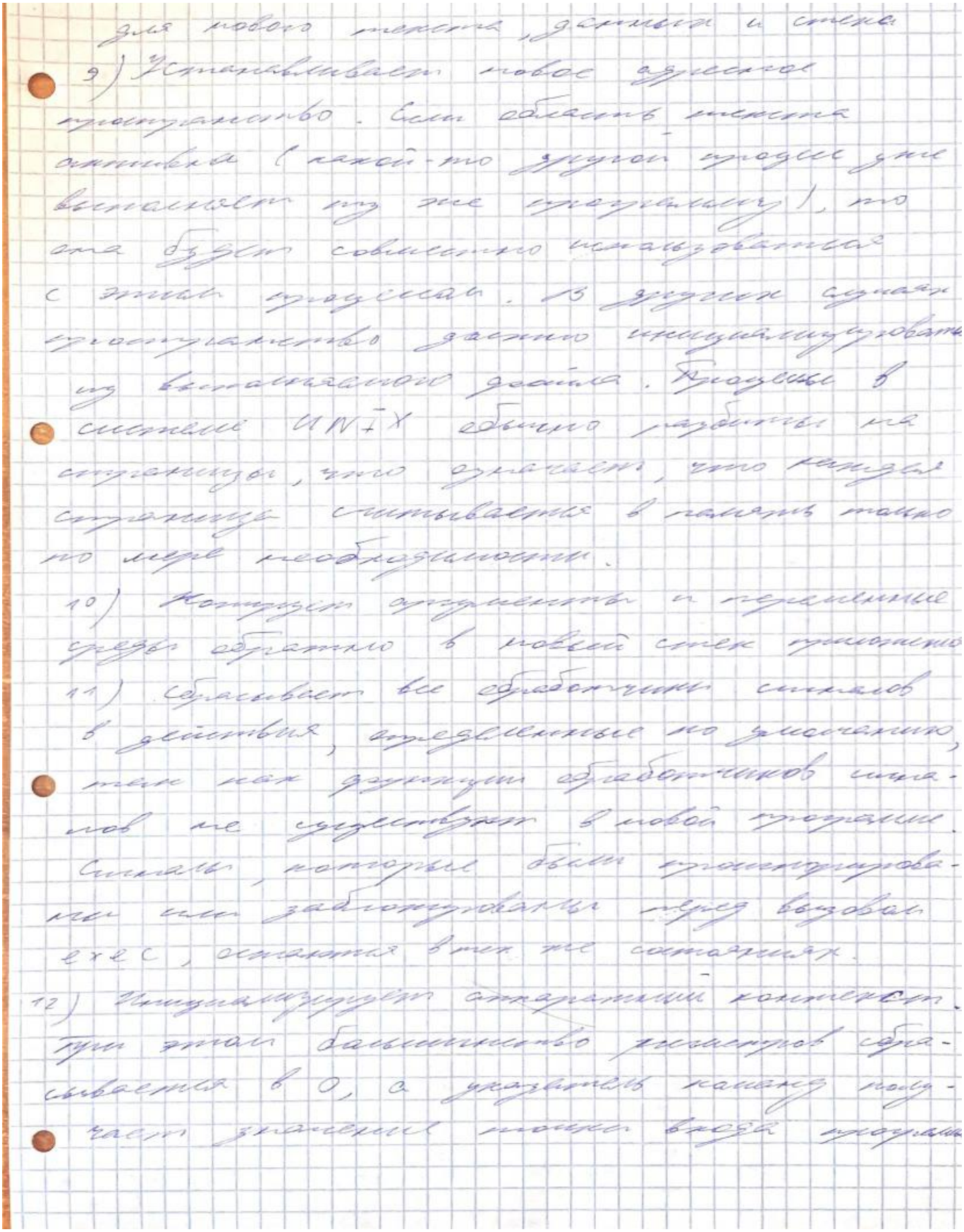


Рис. 10: Очередность при вызове exec().