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 Криков А.В.

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

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



int main()
{
  printf("PARENT: PID = %d, GRP = %d\n", getpid(), getpgrp());
  int childpid1, childpid2;

  if ((childpid1 = fork()) == 1)
  {
    perror("Not forked child 1!\n");
    return 1;
  }
  else if (childpid1 == 0)
  {
    printf("lock: PID = %d, GRP = %d PPID = %d\n", getpid(), getpgrp(),
       getppid());
    sleep(5);
    printf("unlock: PID = %d, GRP = %d PPID = %d\n", getpid(), getpgrp(),
       getppid());
    return 0;
  }

  if ((childpid2 = fork()) == 1)
  {
    perror("Not forked child 2!\n");
    return 1;
  }
  else if (childpid2 == 0)
  {
    printf("lock: PID = %d, GRP = %d PPID = %d\n", getpid(), getpgrp(),
       getppid());
    sleep(10);
    printf("unlock: PID = %d, GRP = %d PPID = %d\n", getpid(), getpgrp(),
       getppid());
    return 0;
  }

  printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
     getpgrp(), childpid1, childpid2);
```

```
38    printf("Parent died!\n\n");
39    return 0;
40 }
```

```
akrik@192 src % ./first.out
PARENT: PID = 4776, GRP = 4776
Parent: id: 4776 pgrp: 4776 child1: 4777 child2: 4778
Parent died!

lock: PID = 4777, GRP = 4776 PPID = 4776
lock: PID = 4778, GRP = 4776 PPID = 4776
akrik@192 src % unlock: PID = 4777, GRP = 4776 PPID = 1
unlock: PID = 4778, GRP = 4776 PPID = 1
```

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()

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/wait.h>
4  #include <stdlib.h>
5
6  void check_status(int status);
7
8  int main()
9  {
10   printf("PARENT: PID = %d, GRP = %d\n\n", getpid(), getpgrp());
11   int childpid1, childpid2;
12
13   if ((childpid1 = fork()) == 1)
14   {
15     perror("Not forked child 1!\n");
16     return 1;
17   }
18   else if (childpid1 == 0)
19   {
20     printf("CHILD_1 : PID = %d, GRP = %d PPID = %d\n", getpid(), getpgrp(),
         getppid());
21     sleep(5);
```

```
22      return 0;
23    }
24
25    if ((childpid2 = fork()) == 1)
26    {
27      perror("Not forked child 2!\n");
28      return 1;
29    }
30    else if (childpid2 == 0)
31    {
32      printf("CHILD_2 : PID = %d, GRP = %d PPID = %d\n", getpid(), getpgrp(),
          getppid());
33      sleep(7);
34      return 0;
35    }
36
37    int status;
38    pid_t childpid;
39
40    childpid = wait(&status);
41    printf("STATUS = %d,  CHILD_PID = %d\n", status, childpid);
42    check_status(status);
43
44    childpid = wait(&status);
45    printf("STATUS = %d,  CHILD_PID = %d\n", status, childpid);
46    check_status(status);
47
48    printf("Parent: id: %d pgrp: %d\n", getpid(), getpgrp());
49    printf("Parent died!\n\n");
50
51    return 0;
52  }
53
54
55  void check_status(int status)
56  {
57    if (WIFEXITED(status))
58    {
59      printf("The child process is completed normally.\n");
60      printf("Child process termination code %d.\n\n", WEXITSTATUS(status));
61      return;
62    }
63
64    if (WIFSIGNALED(status))
65    {
66      printf("The child process terminates with an un-intercepted signal.\n");
67      printf("Signal number %d.\n", WTERMSIG(status));
68      return;
69    }
70
```

```
71    if (WIFSTOPPED(status))
72    {
73      printf("The child process has stopped.\n");
74      printf("Signal number %d.", WSTOPSIG(status));
75    }
76  }
```

```
akrik@192 src % ./second.out
PARENT: PID = 14101, GRP = 14101

CHILD_1 : PID = 14102, GRP = 14101 PPID = 14101
CHILD_2 : PID = 14103, GRP = 14101 PPID = 14101
STATUS = 0,  CHILD_PID = 14102
The child process is completed normally.
Child process termination code 0.

STATUS = 0,  CHILD_PID = 14103
The child process is completed normally.
Child process termination code 0.

Parent: id: 14101 pgrp: 14101
Parent died!
```

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.

execl первого потомка использует программу сортировки массива слиянием.

execl второго потомка использует программу для поиска значения функции с помощью интерполяции полиномом Ньютона.

Параметры передаваемые в execl для первого потомка:
1 - название исполняемого файла
2 - размер входного неотсортированного массива
3..7 - входной неотсортированный массив

Параметры передаваемые в execl для второго потомка:
1 - название исполняемого файла

2 - файл, содержащий в себе промежуточные значения произвольной функции
3 - значение аргумента, для которого нужно найти значение функции
4 - степень интерполяционного полинома

Листинг 3: Вызов функции execl()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>

void check_status(int status);

int main()
{
  printf("PARENT: PID = %d, GRP = %d\n\n", getpid(), getpgrp());
  int childpid1, childpid2;
  int status;
  pid_t childpid;


  if ((childpid1 = fork()) == 1)
  {
    perror("Not forked child 1!\n");
    return 1;
  }
  else if (childpid1 == 0)
  {
    printf("CHILD_1 : PID = %d, GRP = %d PPID = %d\n\n", getpid(), getpgrp()
        , getppid());

     if (execl("merge_sort.out", "5", "19", "12", "14", "11", "42", NULL)
         == -1)
    {
      perror("CHILD_1 cant exec");
      return 1;
    }

    return 0;
  }

  if ((childpid2 = fork()) == 1)
  {
    perror("Not forked child 2!\n");
    return 1;
  }
  else if (childpid2 == 0)
  {
    printf("CHILD_2 : PID = %d, GRP = %d PPID = %d\n\n", getpid(), getpgrp()
        , getppid());

```

```
43          if (execl("newton_interpolation.out", "func.txt", "0.4", "4", NULL)
               == -1)
44      {
45        perror("CHILD_2 cant exec");
46        return 1;
47      }
48
49      return 0;
50    }
51
52    childpid = wait(&status);
53    check_status(status);
54
55    childpid = wait(&status);
56    check_status(status);
57
58    printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
         getpgrp(), childpid1, childpid2);
59    printf("Parent died!\n\n");
60
61    return 0;
62  }
63
64
65  void check_status(int status)
66  {
67    if (WIFEXITED(status))
68    {
69      printf("The child process is completed normally.\n");
70      printf("Child process termination code %d.\n\n", WEXITSTATUS(status));
71      return;
72    }
73
74    if (WIFSIGNALED(status))
75    {
76      printf("The child process terminates with an un-intercepted signal.\n");
77      printf("Signal number %d.\n", WTERMSIG(status));
78      return;
79    }
80
81    if (WIFSTOPPED(status))
82    {
83      printf("The child process has stopped.\n");
84      printf("Signal number %d.", WSTOPSIG(status));
85    }
86  }
```

```
akrik@192 src % ./third.out
PARENT: PID = 4065, GRP = 4065

CHILD_1 : PID = 4066, GRP = 4065 PPID = 4065

CHILD_2 : PID = 4067, GRP = 4065 PPID = 4065

Interpolation result: y = f(0.400) = 0.808858
The child process is completed normally.
Child process termination code 0.

Sorted array = 11, 12, 14, 19, 42
The child process is completed normally.
Child process termination code 0.

Parent: id: 4065 pgrp: 4065 child1: 4066 child2: 4067
Parent died!
```

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

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


void check_status(int status);

int main()
{
  int childpid1, childpid2;
  int fd[2];
  char first_text[57] = "Tyger Tyger, burning bright, In the forests of the
     night";
  char second_text[44] = "The forest paths are muddy, after the rain.";

  if (pipe(fd) == -1)
  {
```

```
19      perror("Cant pipe.\n");
20      return 1;
21    }
22
23    if ((childpid1 = fork()) == -1)
24    {
25      perror("Not forked.\n");
26      return 1;
27    }
28    else if (!childpid1)
29    {
30      close(fd[0]);
31      write(fd[1], first_text, strlen(first_text) + 1);
32
33          printf("First child message send to parent, message: %s\n",
              first_text);
34
35      return 0;
36
37    }
38
39    if ((childpid2 = fork()) == -1)
40    {
41      perror("Not forked.\n");
42      return 1;
43    }
44    else if (!childpid2)
45    {
46      close(fd[0]);
47      write(fd[1], second_text, strlen(second_text) + 1);
48
49          printf("Second child message send to parent, message: %s\n\n",
              second_text);
50
51      return 0;
52    }
53
54    if (childpid1 && childpid2)
55    {
56      char text1[57], text2[44];
57      pid_t childpid;
58      int status;
59
60      close(fd[1]);
61
62      read(fd[0], text1, strlen(first_text) + 1);
63      read(fd[0], text2, strlen(second_text) + 1);
64
65      printf("Text first: %s\n", text1);
66      printf("Text second: %s\n\n", text2);
```

```
67
68      childpid = wait(&status);
69      check_status(status);
70
71      childpid = wait(&status);
72      check_status(status);
73
74      printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
            getpgrp(), childpid1, childpid2);
75        printf("Parent died!\n\n");
76    }
77
78    return 0;
79  }
80
81  void check_status(int status)
82  {
83    if (WIFEXITED(status))
84    {
85      printf("The child process is completed normally.\n");
86      printf("Child process termination code %d.\n\n", WEXITSTATUS(status));
87      return;
88    }
89
90    if (WIFSIGNALED(status))
91    {
92      printf("The child process terminates with an un-intercepted signal.\n");
93      printf("Signal number %d.\n", WTERMSIG(status));
94      return;
95    }
96
97    if (WIFSTOPPED(status))
98    {
99      printf("The child process has stopped.\n");
100     printf("Signal number %d.", WSTOPSIG(status));
101   }
102 }
```

```
akrik@192 src % ./fourth.out
First child message send to parent, message: Tyger Tyger, burning bright, In the forests of the night
Second child message send to parent, message: The forest paths are muddy, after the rain.

Text first: Tyger Tyger, burning bright, In the forests of the night
Text second: The forest paths are muddy, after the rain.

The child process is completed normally.
Child process termination code 0.

The child process is completed normally.
Child process termination code 0.

Parent: id: 1719 pgrp: 1719 child1: 1720 child2: 1721
Parent died!
```

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
#include <stdbool.h>
#include <string.h>


_Bool flag = false;

void check_status(int status);


void catch_sig(int sig_numb)
{
  flag = true;
  printf("catch_sig: %d\n", sig_numb);
}


int main()
{
  signal(SIGINT, catch_sig);

```

```
26      int childpid_1, childpid_2;
27      int fd[2];
28      char text1[57] = "\0", text2[44] = "\0";
29      char first_text[57] = "Tyger Tyger, burning bright, In the forests of
           the night";
30      char second_text[44] = "The forest paths are muddy, after the rain.";
31      pid_t childpid;
32      int status;
33
34      printf("Parent: press CTRL+C if you want to see messages from childs\n\n
           ");
35      sleep(5);
36
37      if (pipe(fd) == -1)
38      {
39        perror("Cant pipe.\n");
40        return 1;
41      }
42
43      if ((childpid_1 = fork()) == -1)
44      {
45        perror("Not forked.\n");
46        return 1;
47      }
48      else if (!childpid_1)
49      {
50        if (flag) {
51          close(fd[0]);
52          write(fd[1], first_text, strlen(first_text) + 1);
53
54          printf("First child message send to parent, message: %s\n",
               first_text);
55        }
56
57        return 0;
58      }
59
60      if ((childpid_2 = fork()) == -1)
61      {
62        perror("Cant fork.\n");
63        return 1;
64      }
65      else if (!childpid_2)
66      {
67        if (flag) {
68          close(fd[0]);
69          write(fd[1], second_text, strlen(second_text) + 1);
70
71          printf("Second child message send to parent, message: %s\n",
               second_text);
```

```
72        }
73
74        return 0;
75      }
76
77      if (childpid_1 && childpid_2)
78      {
79        close(fd[1]);
80
81        read(fd[0], text1, strlen(first_text) + 1);
82        read(fd[0], text2, strlen(second_text) + 1);
83
84        printf("\nText: %s\n", text1);
85        printf("Text: %s\n\n", text2);
86      }
87
88      childpid = wait(&status);
89      check_status(status);
90
91      childpid = wait(&status);
92      check_status(status);
93
94      printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
           getpgrp(), childpid_1, childpid_2);
95      printf("Parent died!\n\n");
96
97      return 0;
98    }
99
100
101
102   void check_status(int status)
103   {
104     if (WIFEXITED(status))
105     {
106       printf("The child process is completed normally.\n");
107       printf("Child process termination code %d.\n\n", WEXITSTATUS(status));
108       return;
109     }
110
111     if (WIFSIGNALED(status))
112     {
113       printf("The child process terminates with an un-intercepted signal.\n"
             );
114       printf("Signal number %d.\n", WTERMSIG(status));
115       return;
116     }
117
118     if (WIFSTOPPED(status))
119     {
```

```
120         printf("The child process has stopped.\n");
121         printf("Signal number %d.", WSTOPSIG(status));
122       }
123     }
```

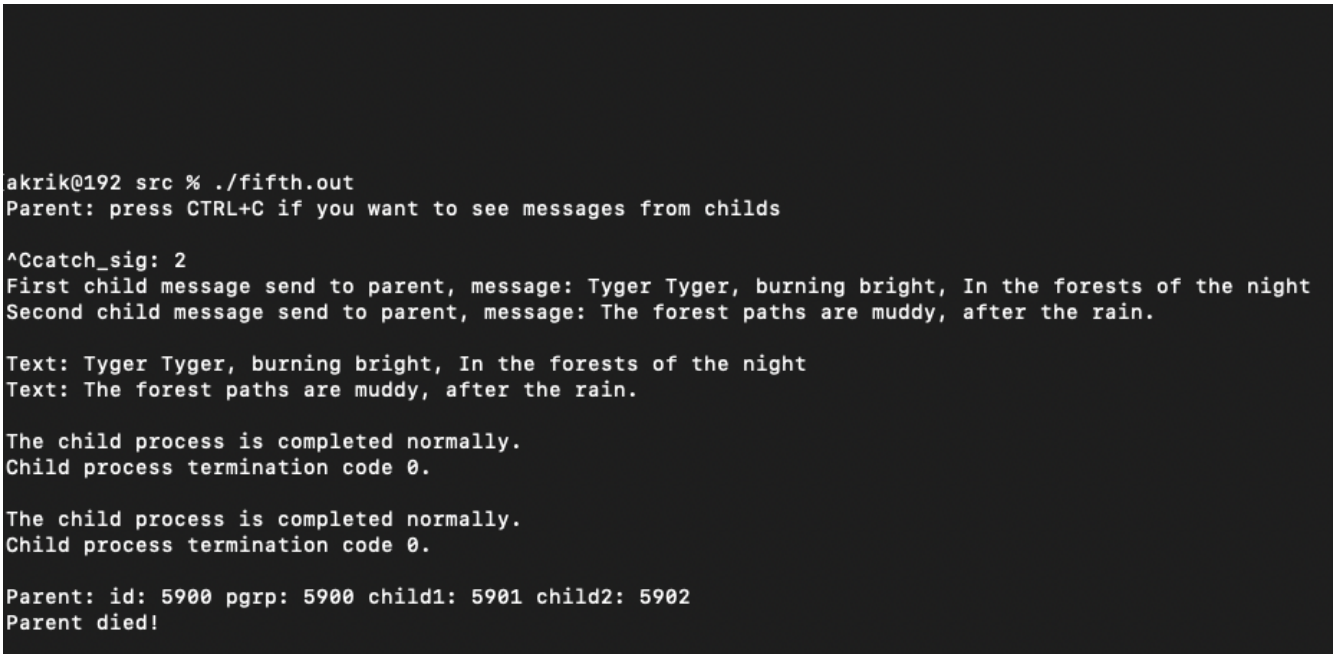


Рис. 5: Демонстрация работы программы, сигнал = SIGINT (задание №5).

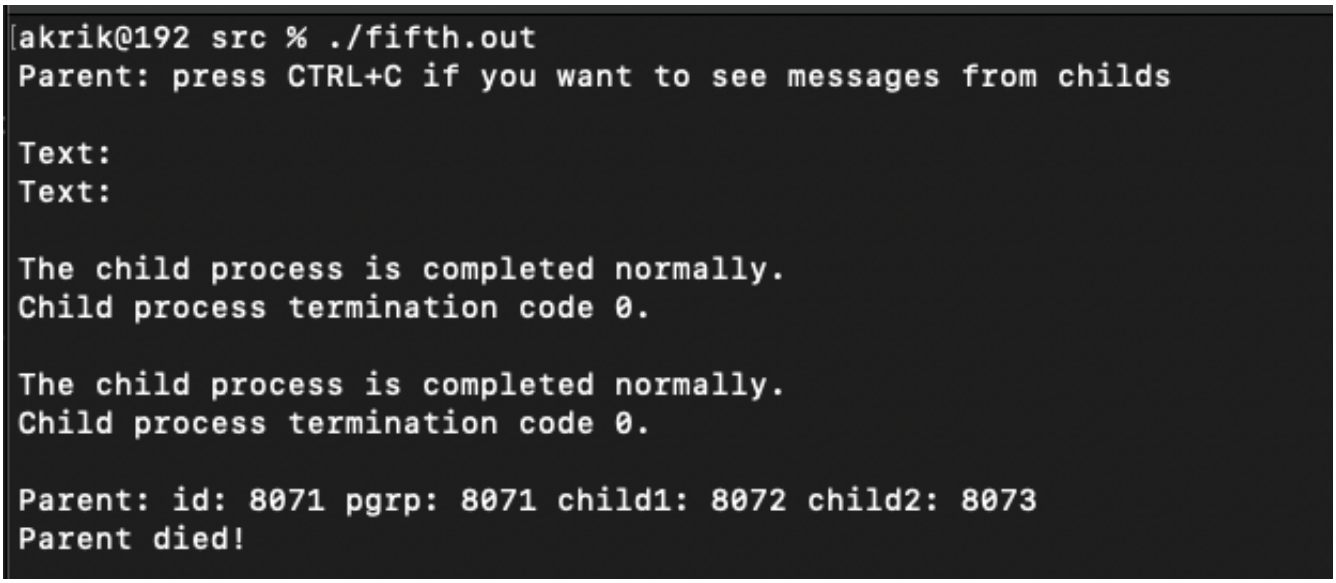


Рис. 6: Демонстрация работы программы, сигнал отсутствует (задание №5).

Дополнительное задание

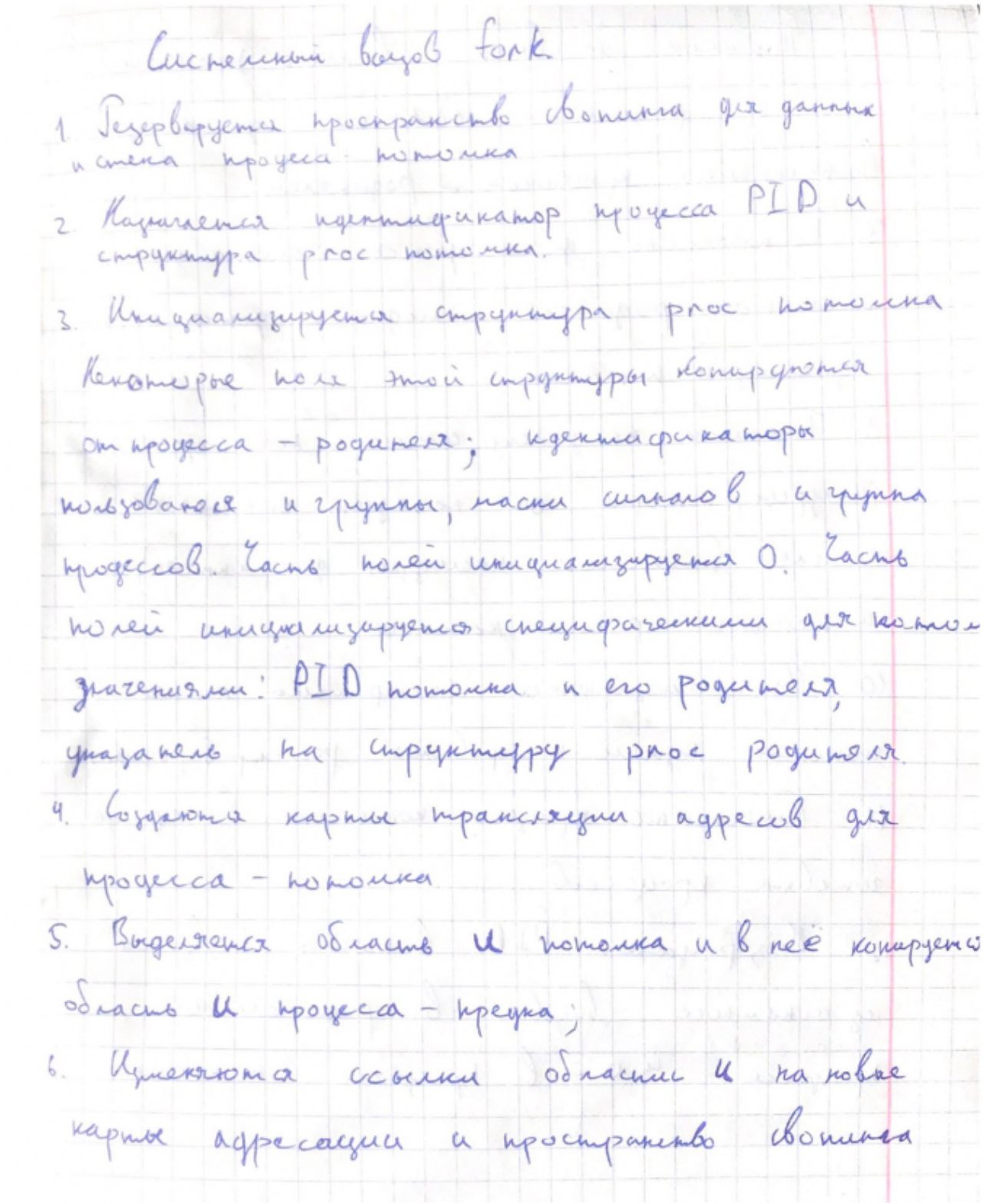


Рис. 7: Очередность действий при вызове fork (1)

7 Потомок добавляется в набор процессов, которые разделяют область кода программы, выполняемой процессом – родителем.

8. Постранично дублируются области данных и стека родителя и модифицируются карты адресации потомка.

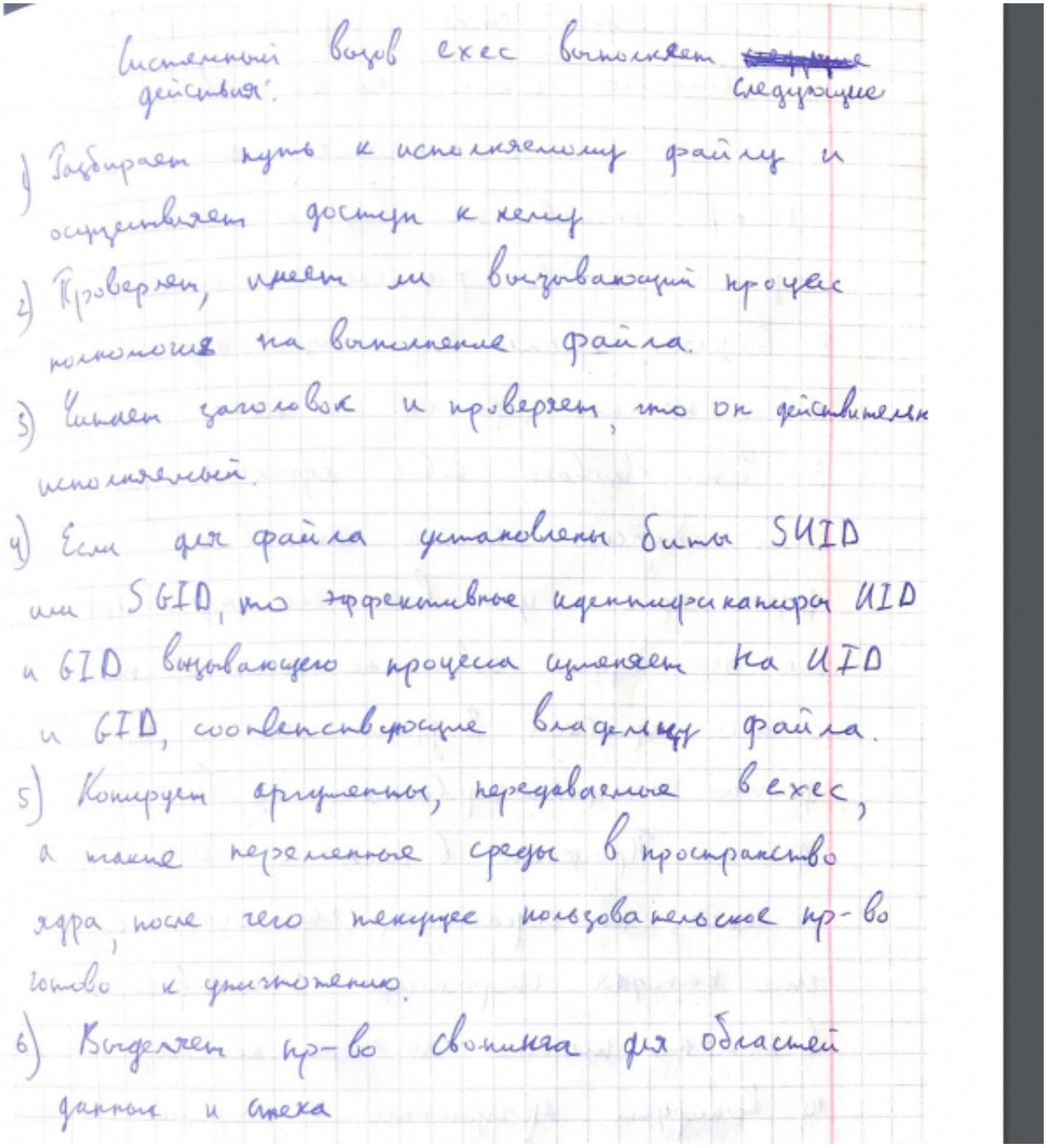
9 Потомок получает ссылки на разделяемые ресурсы, которые он наследует: открытые файлы (потомок наследует дескрипторы) и текущий рабочий каталог;

10 Инициализируется аппаратный контекст потомка путём копирования регистров родителя

11. Поместить процесс-потомок в очередь готовых процессов.

12. Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 - в процессе – потомке.

Рис. 8: Очередность действий при вызове fork (2)


Системный вызов exec выполняет следующие действия:

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 меняет на UID и GID, соответствующие владельцу файла.
5) Копирует аргументы, передаваемые в exec, а также переменные среды в пространство ядра, после чего текущее пользовательское пр-во готово к выполнению.
6) Выделяет пр-во свопинга для областей данных и стека

Рис. 9: Очередность действий при вызове exec (1)

7. Высвобождает старое адресное пр-во и связанное с ним пр-во свопинга. Если же процесс был создан при помощи vfork, производится возврат старого адресного пр-ва родительскому процессу.
8. Выделяет карты трансляции адресов для нового текста, данных и стека.
9. Устанавливает новое адресное пр-во. Если область текста активна (какой-то другой процесс уже выполняет ту же программу), то она будет совместно использоваться с этим процессом. В других случаях пр-во должно инициализироваться из выполняемого файла. Процессы в системе UNIX обычно разбиты на страницы, что означает, что каждая страница считывается в память только по мере необходимости.
10. Копирует аргументы и переменные среды обратно в новый стек приложения.

Рис. 10: Очередность действий при вызове exec (2)

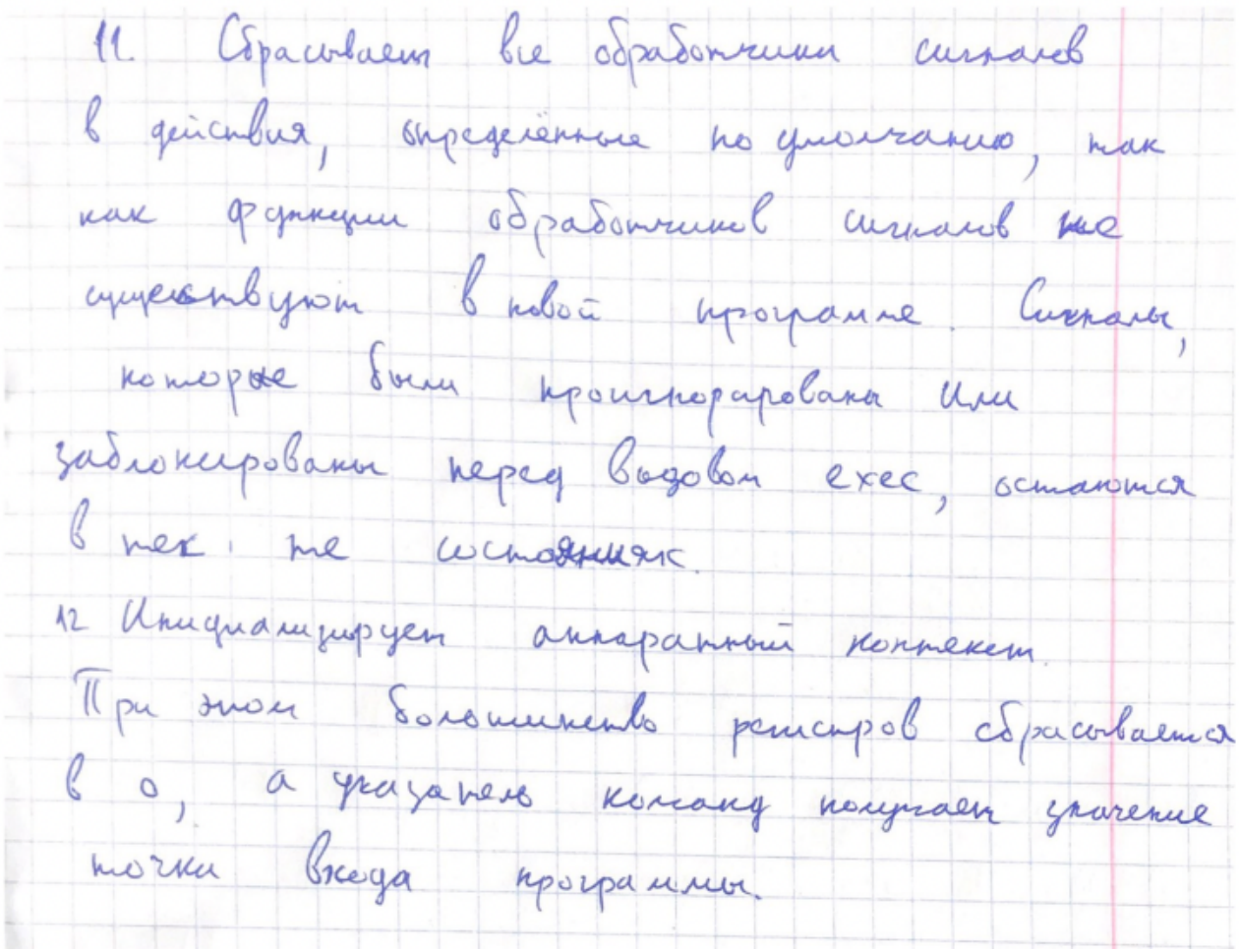


Рис. 11: Очередность действий при вызове exec (3)