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 курсу "Операционные системы"

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

Студент Цветков И. А.

Группа ИУ7-53 Б

Преподаватель Рязанова Н. Ю.

Москва — 2021 г.

Программы лабораторной работы

Задача 1

В программе создаются два процесса потомка. В них вызывается sleep(), чтобы дочерние процессы завершились после завершения процесса-предка. При этом данные процессы станут сиротами.

Листинг 1 – Вызов fork()

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


#define TIME_FOR_SLEEP 3

#define TASK_TEXT "\n=======================\
                   \n      Task 1: fork()     \
                   \n=======================\n\n"


int main()
{
    printf(TASK_TEXT);

    int child1, child2;

    // Порождение процесса-потомка через fork()
    if ((child1 = fork()) == -1)
    {
        perror("Can not fork\n");
        return -1;
    }
    else if (child1 == 0)
    {
        printf("\nBefore sleep: Child 1: pid = %d, ppid = %d, pgrp = %d\n",
            getpid(), getppid(), getpgrp());

        sleep(TIME_FOR_SLEEP);

        printf("\nAfter sleep: Child 1: pid = %d, ppid = %d, pgrp = %d\n",
            getpid(), getppid(), getpgrp());
```

```
32
33          exit(0);
34      }
35
36
37      if ((child2 = fork()) == -1)
38      {
39          perror("Can not fork\n");
40          return -1;
41      }
42      else if (child2 == 0)
43      {
44          printf("\nBefore sleep: Child 2: pid = %d, ppid = %d, pgrp = %d\n",
                getpid(), getppid(), getpgrp());
45
46          sleep(TIME_FOR_SLEEP);
47
48          printf("\nAfter sleep: Child 2: pid = %d, ppid = %d, pgrp = %d\n",
                getpid(), getppid(), getpgrp());
49
50          exit(0);
51      }
52
53
54      printf("\n\nParent: pid = %d, pgrp = %d\n", getpid(), getpgrp());
55
56      return 0;
57  }
```

```
./fork.exe

======================
    Task 1: fork()
======================



Parent: pid = 52421, pgrp = 52421


Before sleep: Child 2: pid = 52423, ppid = 1487, pgrp = 52421
Before sleep: Child 1: pid = 52422, ppid = 1487, pgrp = 52421
amunra23@amunra23:~/studying/sem5/os/os_github/lab_04/src$
After sleep: Child 1: pid = 52422, ppid = 1487, pgrp = 52421

After sleep: Child 2: pid = 52423, ppid = 1487, pgrp = 52421
```

Рисунок 1 – Демонстрация работы программы

Рисунок 2 – Процесс, который усыновил процессы-сироты

Задача 2

Вызов wait() - блокирует родительский процесс (то есть родительскому процессу не выделяется процессорное время) до момента завершения дочернего процесса.

На экран выводятся соответсвующие сообщения.

Листинг 2 – Вызов wait()

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

#include <sys/wait.h>
#include <sys/types.h>

#define TIME_FOR_SLEEP 3


#define TASK_TEXT "\n=======================\
                    \n     Task 2: wait()    \
                    \n=======================\n\n"


void check_status(int status)
{
    if (WIFEXITED(status))
    {
        printf("Дочерний процесс завершен корректно\n");
        printf("Дочерний процесс завершился с кодом: \t%d\n\n",
           WEXITSTATUS(status));

        return;
    }

    if (WIFSIGNALED(status))
    {
        printf("Дочерний процесс завершен неперехватываемым сигналом\n");
        printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));

        return;
    }

    if (WIFSTOPPED(status))
    {
        printf("Дочерний процесс остановлен\n");
        printf("Номер сигнала: \t%d\n\n", WSTOPSIG(status));
```

```
38
39          return;
40      }
41  }
42
43
44  int main()
45  {
46      printf(TASK_TEXT);
47
48      int child1, child2;
49      pid_t child_pid;
50      int status;
51
52      if ((child1 = fork()) == -1)
53      {
54          perror("Can not fork\n");
55          return -1;
56      }
57      else if (child1 == 0)
58      {
59          sleep(TIME_FOR_SLEEP);
60
61          printf("\nChild 1: pid = %d, ppid = %d, pgrp = %d\n", getpid(),
                getppid(), getpgrp());
62
63          exit(0);
64      }
65
66      if ((child2 = fork()) == -1)
67      {
68          perror("Can not fork\n");
69          return -1;
70      }
71      else if (child2 == 0)
72      {
73          sleep(TIME_FOR_SLEEP);
74
75          printf("\nChild 2: pid = %d, ppid = %d, pgrp = %d\n\n", getpid(),
                getppid(), getpgrp());
76
77          exit(0);
78      }
79
80      child_pid = wait(&status);
81      printf("Process status: %d, child pid = %d\n", status, child_pid);
82      check_status(status);
83
```

```
84      child_pid = wait(&status);
85      printf("Process status: %d, child pid = %d\n", status, child_pid);
86      check_status(status);
87
88      printf("\nParent: pid = %d, pgrp = %d\nChild1 = %d, Child2 = %d\n",
            getpid(), getpgrp(), child1, child2);
89
90      return 0;
91  }
```

```
amunra23@amunra23:~/studying/sem5/os/os_github/lab_04/src$ ./wait.exe

======================
     Task 2: wait()
======================



Child 2: pid = 50702, ppid = 50700, pgrp = 50700

Child 1: pid = 50701, ppid = 50700, pgrp = 50700
Process status: 0, child pid = 50701
Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0

Process status: 0, child pid = 50702
Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0


Parent: pid = 50700, pgrp = 50700
Child1 = 50701, Child2 = 50702
```

Рисунок 3 – Демонстрация работы программы

Задача 3

Процессы-потомки переходят на выполнение других программ: первый процесс-потомок выполняет программу, которая производит некоторую работу с графом (лабораторная работа по Типам и Структурам данных), а второй процесс-потомок выполняет программу, сортирующую массив целых чисел методом быстрой сортировки (QuickSort).

На экран выводятся соответсвующие сообщения.

Листинг 3 – Вызов execlp()

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

#include <sys/wait.h>
#include <sys/types.h>


#define TIME_FOR_SLEEP 3

#define TASK_TEXT "\n=======================\
                   \n     Task 3: exec()    \
                   \n=======================\n\n"


void check_status(int status)
{
    if (WIFEXITED(status))
    {
        printf("Дочерний процесс завершен корректно\n");
        printf("Дочерний процесс завершился с кодом: \t%d\n\n",
           WEXITSTATUS(status));

        return;
    }

    if (WIFSIGNALED(status))
    {
        printf("Дочерний процесс завершен неперехватываемым сигналом\n");
        printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));

        return;
    }

    if (WIFSTOPPED(status))
```

```
35      {
36          printf("Дочерний процесс остановлен\n");
37          printf("Номер сигнала: \t%d\n\n", WSTOPSIG(status));
38
39          return;
40      }
41  }
42
43
44  int main()
45  {
46      printf(TASK_TEXT);
47
48      int child1, child2;
49      pid_t child_pid;
50      int status;
51
52      if ((child1 = fork()) == -1)
53      {
54          perror("Can not fork\n");
55          return -1;
56      }
57      else if (child1 == 0)
58      {
59          if (execlp("./tads_graph/tads_graph.exe", "./tads_graph.exe",
                "./tads_graph/input.txt", NULL) == -1)
60          {
61              printf("\nError: Child 1 can not execute exec()\n");
62
63              exit(0);
64          }
65
66          exit(0);
67      }
68
69      if ((child2 = fork()) == -1)
70      {
71          perror("Can not fork\n");
72          return -1;
73      }
74      else if (child2 == 0)
75      {
76          if (execlp("./qsort/qsort.exe", "./qsort.exe", "./qsort/input.txt",
                NULL) == -1)
77          {
78              printf("\nError: Child 2 can not execute exec()\n");
79
80              exit(0);
```

```
81          }
82
83          exit(0);
84      }
85
86      child_pid = wait(&status);
87      printf("\n\nProcess status: %d, child pid = %d\n", status, child_pid);
88      check_status(status);
89
90      child_pid = wait(&status);
91      printf("Process status: %d, child pid = %d\n", status, child_pid);
92      check_status(status);
93
94      printf("\nParent: pid = %d, pgrp = %d\nChild1 = %d, Child2 = %d\n",
            getpid(), getpgrp(), child1, child2);
95
96      return 0;
97  }
```

```
     Task 3: exec()
======================


Child 1: pid = 94244, ppid = 94243, pgrp = 94236

Child 2: pid = 94245, ppid = 94243, pgrp = 94236

Количество городов (вершин графа): 3

Длина пути:
из 1 в 2: 1
из 1 в 3: 1
из 2 в 3: 3

Город, из которого будет производиться поиск: 2
Максимальная длина пути: 2

Кратчайшие пути в город 2 (не превышющие 2):
Из города 1 : 1 -> 2
Из города 2 : 2
Из города 3 : 3 -> 1 -> 2

Найдено построение графа



Process status: 0, child pid = 94244
Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0


Сортировка массива методом QuickSort

Изначальный массив: 5 1 -3 9 23 -1 0 23 -7 11

Отсортированный массив: -7 -3 -1 0 1 5 9 11 23 23

Process status: 0, child pid = 94245
Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0
```

Рисунок 4 – Демонстрация работы программы

Задача 4

Процессы-потомки и процесс-предок обмениваются сообщениями через неименованный программный канал. На экран выводятся соответсвующие сообщения.

Листинг 4 – Использование pipe()

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

#include <sys/wait.h>
#include <sys/types.h>

#include <string.h>

#define TIME_FOR_SLEEP 3

#define TASK_TEXT "\n=======================\
                   \n      Task 4: pipe()     \
                   \n=======================\n\n"


#define TEXT_CHILD_1 "Everybody wants to belive in miracles\n"
#define TEXT_CHILD_2 "Testing message\n"
#define TEXT_BUF 55



void check_status(int status)
{
    if (WIFEXITED(status))
    {
        printf("Дочерний процесс завершен корректно\n");
        printf("Дочерний процесс завершился с кодом: \t%d\n\n",
           WEXITSTATUS(status));

        return;
    }

    if (WIFSIGNALED(status))
    {
        printf("Дочерний процесс завершен неперехватываемым сигналом\n");
        printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));

        return;
```

```
39      }
40
41      if (WIFSTOPPED(status))
42      {
43          printf("Дочерний процесс остановлен\n");
44          printf("Номер сигнала: \t%d\n\n", WSTOPSIG(status));
45
46          return;
47      }
48  }
49
50
51  int main()
52  {
53      printf(TASK_TEXT);
54
55      int child1, child2;
56
57      int fd[2];
58      pid_t child_pid;
59      int status;
60
61      if (pipe(fd) == -1)
62      {
63          perror("Can not pipe\n");
64          return -1;
65      }
66
67      if ((child1 = fork()) == -1)
68      {
69          perror("Can not fork\n");
70          return -1;
71      }
72      else if (child1 == 0)
73      {
74          close(fd[0]);
75          write(fd[1], TEXT_CHILD_1, strlen(TEXT_CHILD_1) + 1);
76          exit(0);
77      }
78
79      if ((child2 = fork()) == -1)
80      {
81          perror("Can not fork\n");
82          return -1;
83      }
84      else if (child2 == 0)
85      {
86          close(fd[0]);
```

```
87          write(fd[1], TEXT_CHILD_2, strlen(TEXT_CHILD_2) + 1);
88          exit(0);
89      }
90
91
92      if (child1 && child2)
93      {
94          char text1[TEXT_BUF];
95          char text2[TEXT_BUF];
96
97          close(fd[1]);
98
99          read(fd[0], text1, strlen(TEXT_CHILD_1) + 1);
100         read(fd[0], text2, strlen(TEXT_CHILD_2) + 1);
101
102         printf("Result: %s", text1);
103         printf("Result: %s\n\n", text2);
104
105         child_pid = wait(&status);
106         check_status(status);
107
108         child_pid = wait(&status);
109         check_status(status);
110     }
111
112     return 0;
113 }
```

```
amunra23@amunra23:~/studying/sem5/os/os_github/lab_04/src$ make 4
gcc task4.c -o pipe.exe  && ./pipe.exe

========================
     Task 4: pipe()
========================

Result: Everybody wants to belive in miracles
Result: Testing message


Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0

Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0
```

Рисунок 5 – Демонстрация работы программы

Задача 5

Процессы-потомки и процесс-предок обмениваются сообщениями через неименованный программный канал. Используя сигналы, меняем ход выполнения программы.

На экран выводятся соответсвующие сообщения.

Листинг 5 – Использование signal()

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <signal.h>
6
7  #include <sys/wait.h>
8  #include <sys/types.h>
9
10 #include <string.h>
11
12
13 #define TIME_FOR_SLEEP 5
14
15 #define TASK_TEXT "\n=======================\
16                    \n     Task 5: signal()    \
17                    \n=======================\n\n"
18
19
20 #define SIGNAL_TEXT "Зажмите: CTRL+\\ - вывести сообщения от потомков\n\n"
21
22
23 #define TEXT_CHILD_1 "Everybody wants to belive in miracles\n"
24 #define TEXT_CHILD_2 "Testing message\n"
25 #define TEXT_BUF 55
26 flag = 0
27
28
29
30 void check_status(int status)
31 {
32     if (WIFEXITED(status))
33     {
34         printf("Дочерний процесс завершен корректно\n");
35         printf("Дочерний процесс завершился с кодом: \t%d\n\n",
              WEXITSTATUS(status));
36
37         return;
```

```
38      }
39
40      if (WIFSIGNALED(status))
41      {
42          printf("Дочерний процесс завершен неперехватываемым сигналом\n");
43          printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));
44
45          return;
46      }
47
48      if (WIFSTOPPED(status))
49      {
50          printf("Дочерний процесс остановлен\n");
51          printf("Номер сигнала: \t%d\n\n", WSTOPSIG(status));
52
53          return;
54      }
55  }
56
57
58  void catch_ctrlslash(int signal)
59  {
60      flag = 1;
61
62      printf("\nПойманный сигнал = %d\n", signal);
63  }
64
65  int main()
66  {
67      printf(TASK_TEXT);
68
69      char text1[TEXT_BUF] = "\0";
70      char text2[TEXT_BUF] = "\0";
71
72      pid_t child_pid;
73      int status;
74
75      int child1, child2;
76      int fd[2];
77
78      signal(SIGQUIT, catch_ctrlslash);
79
80      printf(SIGNAL_TEXT);
81
82      sleep(TIME_FOR_SLEEP);
83
84      if (pipe(fd) == -1)
85      {
```

```
86          perror("Can not pipe\n");
87          return -1;
88      }
89
90      if ((child1 = fork()) == -1)
91      {
92          perror("Can not fork\n");
93          return -1;
94      }
95      else if (child1 == 0)
96      {
97          if (flag == 1)
98          {
99              close(fd[0]);
100             write(fd[1], TEXT_CHILD_1, strlen(TEXT_CHILD_1) + 1);
101         }
102
103         exit(0);
104     }
105
106     if ((child2 = fork()) == -1)
107     {
108         perror("Can not fork\n");
109         return -1;
110     }
111     else if (child2 == 0)
112     {
113         if (flag == 1)
114         {
115             close(fd[0]);
116             write(fd[1], TEXT_CHILD_2, strlen(TEXT_CHILD_2) + 1);
117         }
118
119         exit(0);
120     }
121
122     if (child1 && child2)
123     {
124         close(fd[1]);
125
126         read(fd[0], text1, strlen(TEXT_CHILD_1) + 1);
127         read(fd[0], text2, strlen(TEXT_CHILD_2) + 1);
128
129         printf("\nResult: %s\n", text1);
130         printf("Result: %s\n\n", text2);
131
132         child_pid = wait(&status);
133         check_status(status);
```

```
134
135         child_pid = wait(&status);
136         check_status(status);
137     }
138
139     return 0;
140 }
```

```
amunra23@amunra23:~/studying/sem5/os/os_github/lab_04/src$ ./signal.exe

======================
     Task 5: signal()
======================

Зажмите: CTRL+\ - вывести сообщения от потомков

^\
Пойманный сигнал = 3

Result: Everybody wants to belive in miracles

Result: Testing message


Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0

Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0
```

Рисунок 6 – Демонстрация работы программы - вызван сигнал

```
amunra23@amunra23:~/studying/sem5/os/os_github/lab_04/src$ ./signal.exe

======================
     Task 5: signal()
======================

Зажмите: CTRL+\ - вывести сообщения от потомков


Result:
Result:

Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0

Дочерний процесс завершен корректно
Дочерний процесс завершился с кодом:    0
```

Рисунок 7 – Демонстрация работы программы - сигнал не вызван

Последовательность действий при вызове fork() и exec()

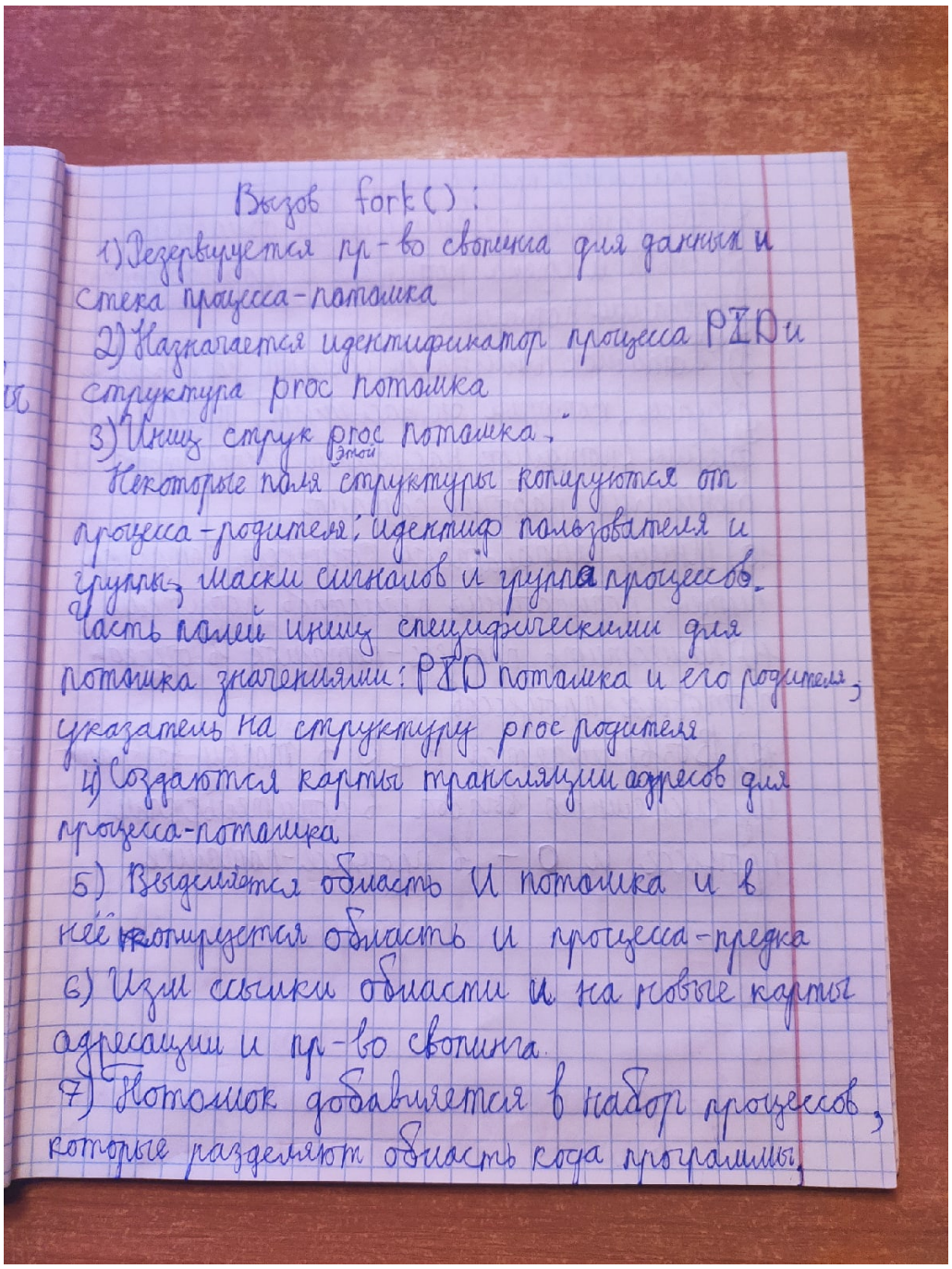

Рисунок 8 – Вызов fork - 1

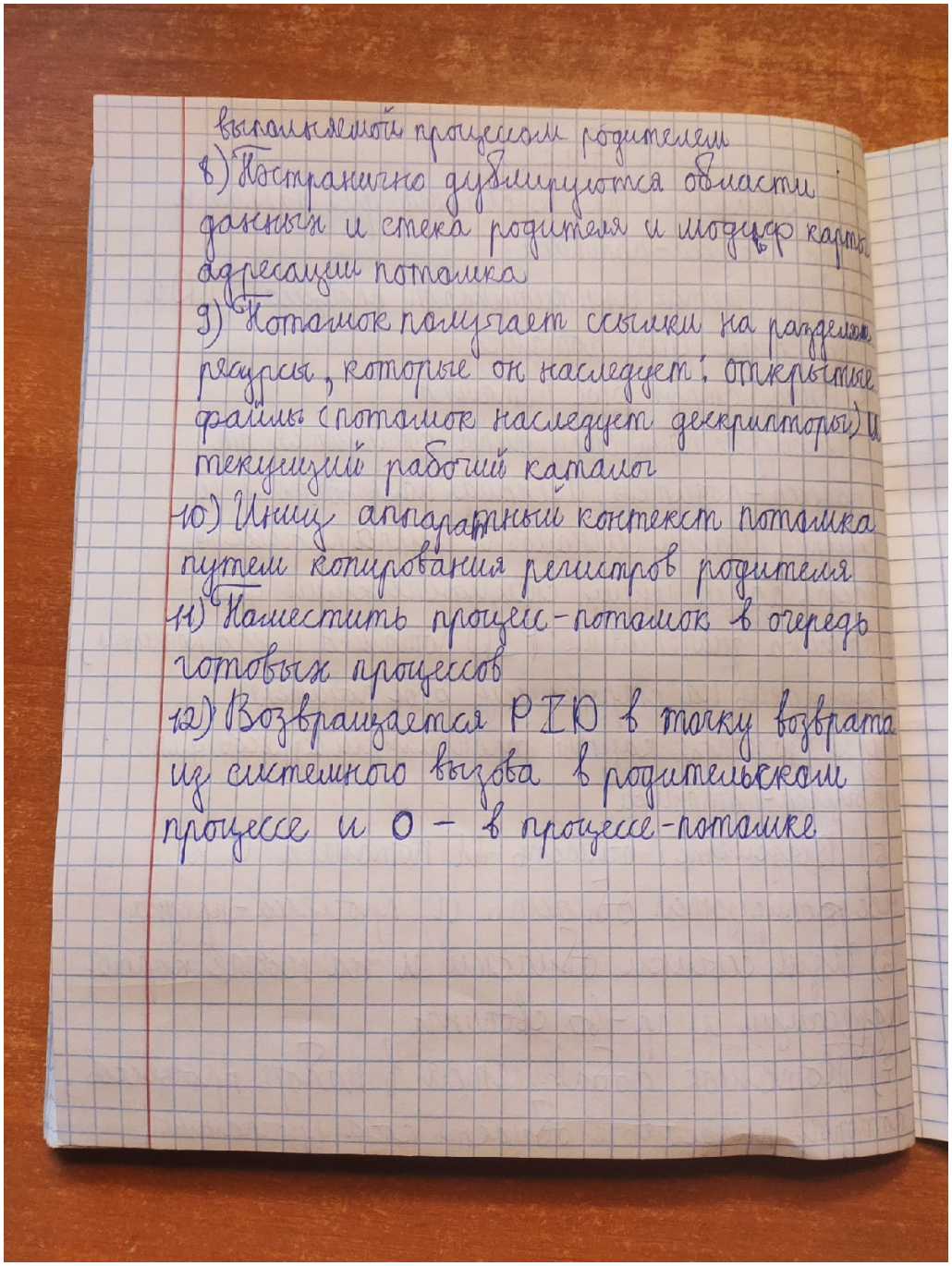

Рисунок 9 – Вызов fork - 2

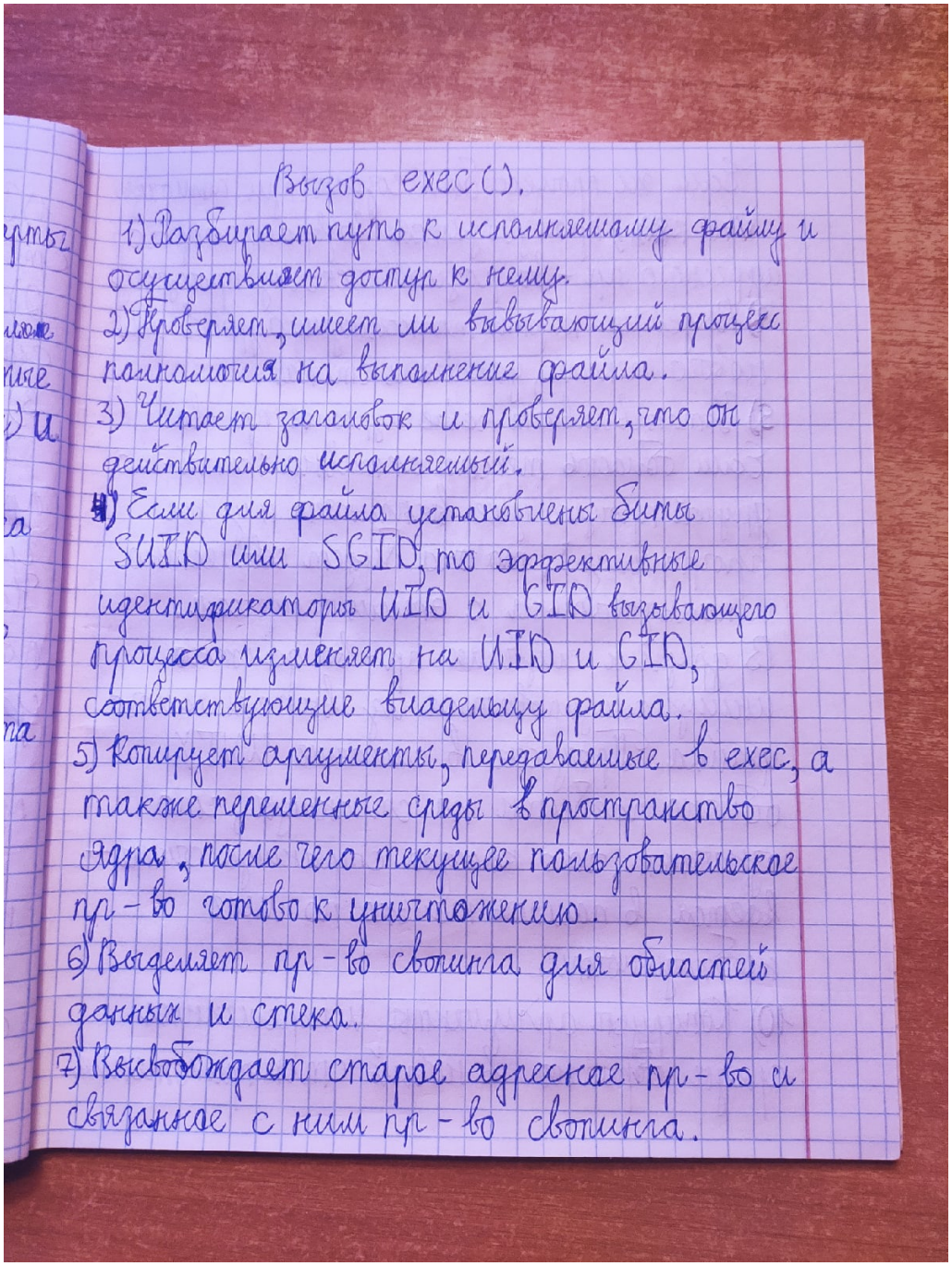

Рисунок 10 – Вызов exec - 1

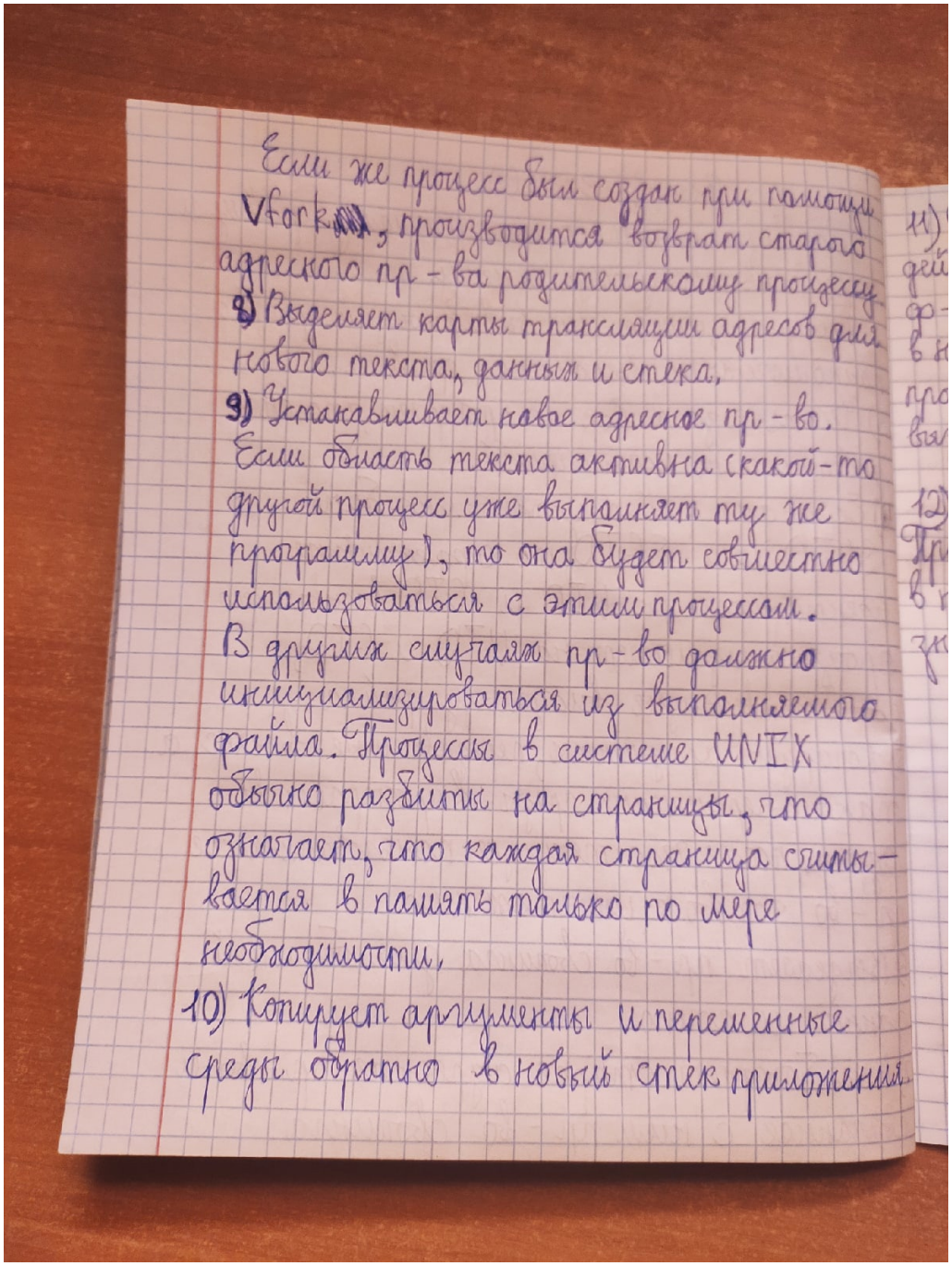

Рисунок 11 – Вызов exec - 2

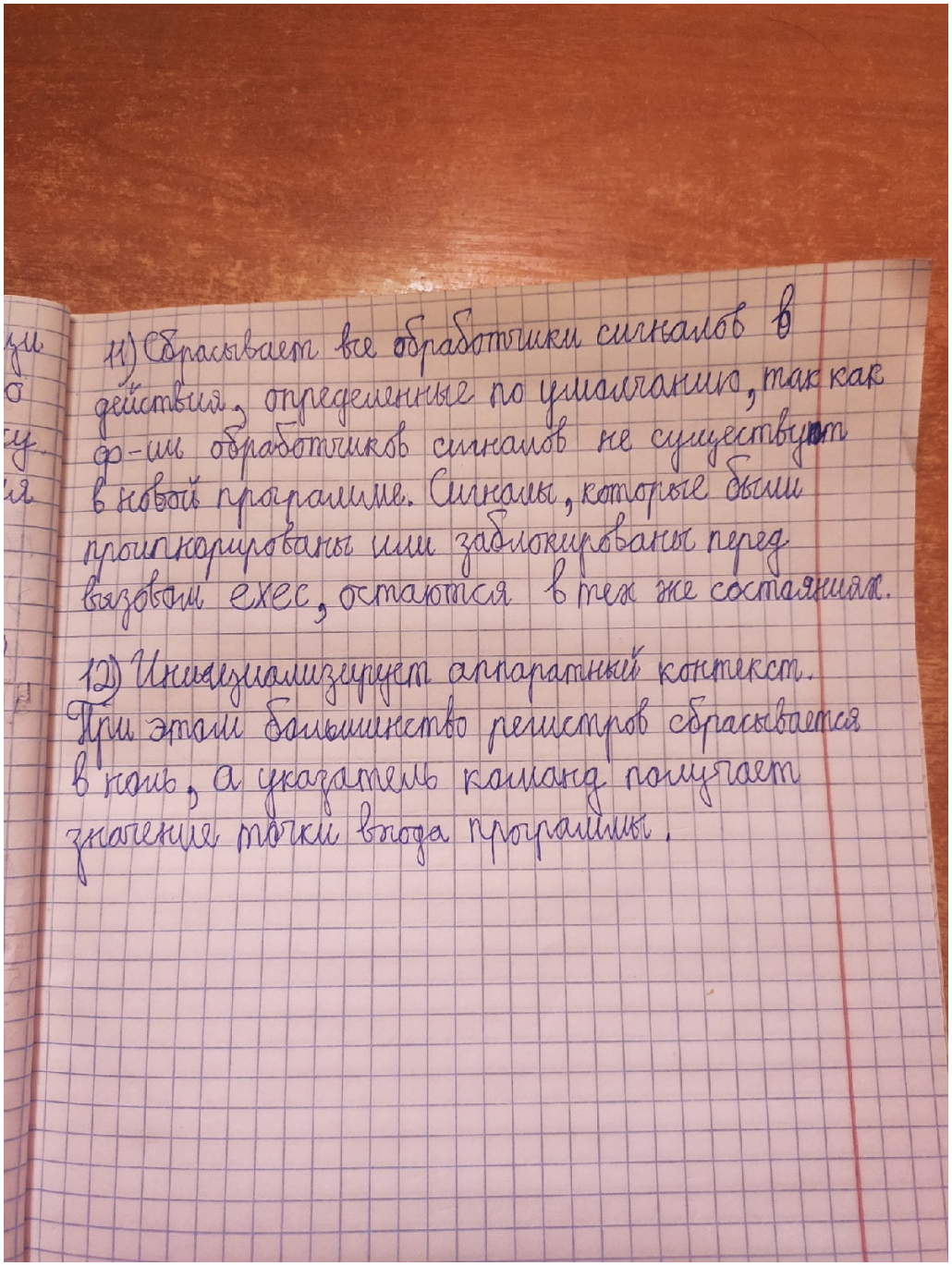

Рисунок 12 – Вызов exec - 3