



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 Хамзина Р. Р.

Группа ИУ7-53Б

Преподаватель Рязанова Н. Ю.

Москва — 2021 г.

Задание 1

При помощи системного вызова `fork()` создается два процесса-потомка. Для того, чтобы они завершились после процесса-предка, в них вызывается `sleep()`. При этом процессы-потомки становятся сиротами. Их усыновляет процесс-посредник `systemd -user`, который является потомком процесса с идентификатором 1.

Листинг 1 – Создание процессов-сирот

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5
6 #define FORK_ERROR -1
7 #define ERROR      1
8 #define OK         0
9
10 #define PAUSE      4
11
12 #define TASK        "\n<<<<< Task 1 : Creating orphan processes
    >>>>>\n\n"
13
14
15 int main(void)
16 {
17     pid_t child_pid1, child_pid2;
18
19     if ((child_pid1 = fork()) == FORK_ERROR)
20     {
21         perror("\nCan't fork child 1.\n");
22         exit(ERROR);
23     }
24     else if (child_pid1 == 0)
25     {
26         printf("\nBEFORE: \
27             \nChild 1: PID = %d, PPID = %d, GPID = %d.\n",
                getpid(), getppid(), getpgrp());
```

```

28
29     sleep (PAUSE);
30
31     printf("\nAFTER: \
32         \nChild 1: PID = %d, PPID = %d, GPID = %d.\n",
            getpid(), getppid(), getpgrp());
33
34     exit (OK);
35 }
36 else
37 {
38     printf("\nParent: PID = %d, GPID = %d, child 1 PID =
        %d.\n", getpid(), getpgrp(), child_pid1);
39 }
40
41
42 if ((child_pid2 = fork()) == FORK_ERROR)
43 {
44     perror("\nCan't fork child 2.\n");
45     exit (ERROR);
46 }
47 else if (child_pid2 == 0)
48 {
49     printf("\nBEFORE: \
50         \nChild 2: PID = %d, PPID = %d, GPID = %d.\n",
            getpid(), getppid(), getpgrp());
51
52     sleep (PAUSE);
53
54     printf("\nAFTER: \
55         \nChild 2: PID = %d, PPID = %d, GPID = %d.\n",
            getpid(), getppid(), getpgrp());
56
57     exit (OK);
58 }
59 else
60 {
61     printf("\nParent: PID = %d, GPID = %d, child 2 PID =
        %d.\n", getpid(), getpgrp(), child_pid2);
62 }
63

```

```

64     return OK;
65 }

```

```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ ./main_01.exe

<<<<< Task 1 : Creating orphan processes >>>>>

Parent: PID = 55759, GPID = 55759, child 1 PID = 55760.

BEFORE:
Child 1: PID = 55760, PPID = 55759, GPID = 55759.

Parent: PID = 55759, GPID = 55759, child 2 PID = 55761.

BEFORE:
Child 2: PID = 55761, PPID = 55759, GPID = 55759.
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$
AFTER:
Child 1: PID = 55760, PPID = 1505, GPID = 55759.

AFTER:
Child 2: PID = 55761, PPID = 1505, GPID = 55759.

```

Рисунок 1 – Демонстрация работы программы

1	1484	1484	1484 ?	-1 Ssl	120	0:00	/usr/bin/who
1	1487	1487	1487 ?	-1 Ss	116	0:02	/usr/sbin/ker
1	1496	1496	1496 ?	-1 Ss	116	0:02	/usr/sbin/ker
1	1505	1505	1505 ?	-1 Ss	1000	0:01	/lib/systemd/
1505	1506	1505	1505 ?	-1 S	1000	0:00	(sd-pam)
1505	1511	1511	1511 ?	-1 S<sl	1000	24:11	/usr/bin/puls
1505	1513	1513	1513 ?	-1 Ssl	1000	0:00	/usr/libexec/

Рисунок 2 – Процесс, усыновивший процессы-сироты

Задание 2

Системный вызов `wait()` блокирует процесс-предок, поэтому он ждет завершения процессов-потомков. Процесс-предок анализирует коды завершения процессов-потомков.

Листинг 2 – Системный вызов wait()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6
7 #define FORK_ERROR -1
8 #define ERROR      1
9 #define OK         0
10
11 #define PAUSE      4
12
13 #define TASK        "\n<<<<< Task 2 : Parent is waiting childs
    with wait() >>>>>\n\n"
14
15
16 void check_status(int status)
17 {
18     if (WIFEXITED(status))
19     {
20         printf("Child exited correctly with code %d.\n",
            WEXITSTATUS(status));
21
22         return;
23     }
24     else if (WIFSIGNALED(status))
25     {
26         printf("Child exited with non-interceptable signal
            %d.\n", WTERMSIG(status));
27
28         return;
29     }
30     else if (WIFSTOPPED(status))
31     {
32         printf("Child stopped with signal %d.\n",
            WSTOPSIG(status));
33
34         return;
35     }
36 }
```

```

37
38
39 int main(void)
40 {
41     printf(TASK);
42
43     pid_t child_pid1, child_pid2, child_pid;
44     int status;
45
46     if ((child_pid1 = fork()) == FORK_ERROR)
47     {
48         perror("\nCan't fork child 1.\n");
49         exit(ERROR);
50     }
51     else if (child_pid1 == 0)
52     {
53         printf("\nBEFORE: \
54             \nChild 1: PID = %d, PPID = %d, GPID = %d.\n",
55                 getpid(), getppid(), getpgrp());
56
57         sleep(PAUSE);
58
59         printf("\nAFTER: \
60             \nChild 1: PID = %d, PPID = %d, GPID = %d.\n",
61                 getpid(), getppid(), getpgrp());
62
63         exit(OK);
64     }
65     else
66     {
67         child_pid = wait(&status);
68         printf("\nChild 1 has fihished: PID = %d, status =
69             %d.\n", child_pid, status);
70
71         printf("\nParent: PID = %d, GPID = %d, child 1 PID =
72             %d.\n", getpid(), getpgrp(), child_pid1);
73         check_status(status);
74     }
75
76     if ((child_pid2 = fork()) == FORK_ERROR)

```

```

74 {
75     perror("\nCan't fork child 2.\n");
76     exit(ERROR);
77 }
78 else if (child_pid2 == 0)
79 {
80     printf("\n\n\nBEFORE: \
81         \nChild 2: PID = %d, PPID = %d, GPID = %d.\n",
82             getpid(), getppid(), getpgrp());
83
84     sleep(PAUSE);
85
86     printf("\n\nAFTER: \
87         \nChild 2: PID = %d, PPID = %d, GPID = %d.\n",
88             getpid(), getppid(), getpgrp());
89
90     exit(OK);
91 }
92 else
93 {
94     child_pid = wait(&status);
95     printf("\nChild 2 has finished: PID = %d, status =
96         %d.\n", child_pid, status);
97
98     printf("\nParent: PID = %d, GPID = %d, child 2 PID =
99         %d.\n", getpid(), getpgrp(), child_pid1);
100     check_status(status);
101 }
102
103 return OK;
104 }

```

```
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ ./main_02.exe

<<<<< Task 2 : Parent is waiting childs with wait() >>>>>

BEFORE:
Child 1: PID = 56422, PPID = 56421, GPID = 56421.

AFTER:
Child 1: PID = 56422, PPID = 56421, GPID = 56421.

Child 1 has fihished: PID = 56422, status = 0.

Parent: PID = 56421, GPID = 56421, child 1 PID = 56422.
Child exited correctly with code 0.


BEFORE:
Child 2: PID = 56424, PPID = 56421, GPID = 56421.

AFTER:
Child 2: PID = 56424, PPID = 56421, GPID = 56421.

Child 2 has fihished: PID = 56424, status = 0.

Parent: PID = 56421, GPID = 56421, child 2 PID = 56422.
Child exited correctly with code 0.
```

Рисунок 3 – Демонстрация работы программы

Задание 3

Процессы-потомки переходят на выполнение следующих программ:

- в первой программе (./sort.exe) выполняется чтение массива целых чисел из файла (array.txt), сортировка массива и его вывод на экран (лабораторная работа по курсу программирования на языке C);
- во второй программе (./palindrome.exe) выполняется чтение последовательности символов из файла (./string) и вывод сообщения о том, является ли введенная последовательность палиндромом (лабораторная работа по курсу программирования на языке C).

Программы передаются системному вызову `execvp()` в качестве параметра.

Листинг 3 – Системный вызов execlp

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6
7 #define FORK_ERROR -1
8 #define EXEC_ERROR -1
9 #define ERROR      1
10 #define OK         0
11
12 #define TASK        "\n<<<<< Task 3 : Children do other programs
    with execlp() >>>>>\n\n"
13
14
15 void check_status(int status)
16 {
17     if (WIFEXITED(status))
18     {
19         printf("Child exited correctly with code %d.\n",
20             WEXITSTATUS(status));
21
22         return;
23     }
24     else if (WIFSIGNALED(status))
25     {
26         printf("Child exited with non-interceptable signal
27             %d.\n", WTERMSIG(status));
28
29         return;
30     }
31     else if (WIFSTOPPED(status))
32     {
33         printf("Child stopped with signal %d.\n",
34             WSTOPSIG(status));
35
36         return;
37     }
38 }
```

```

37
38 int main(void)
39 {
40     printf(TASK);
41
42     pid_t child_pid1, child_pid2, child_pid;
43     int status;
44
45     if ((child_pid1 = fork()) == FORK_ERROR)
46     {
47         perror("\nCan't fork child 1.\n");
48         exit(ERROR);
49     }
50     else if (child_pid1 == 0)
51     {
52         printf("\nChild 1 START: PID = %d, PPID = %d, GPID =
53             %d.\n", getpid(), getppid(), getpgrp());
54         printf("\n- to sort array\n");
55
56         if (execlp("./sort.exe", "./sort.exe", "array.txt", NULL)
57             == EXEC_ERROR)
58         {
59             printf("\nERROR: child 1 can not execute exec().\n");
60             exit(ERROR);
61         }
62         exit(OK);
63     }
64     else
65     {
66         child_pid = wait(&status);
67         printf("\nChild 1 END: PID = %d, status = %d.\n",
68             child_pid, status);
69
70         printf("\nParent: PID = %d, GPID = %d, child 1 PID =
71             %d.\n", getpid(), getpgrp(), child_pid1);
72         check_status(status);
73     }

```

```

74     if ((child_pid2 = fork()) == FORK_ERROR)
75     {
76         perror("\nCan't fork child 2.\n");
77         exit(ERROR);
78     }
79     else if (child_pid2 == 0)
80     {
81         printf("\n\n\nChild 2 START: PID = %d, PPID = %d, GPID =
            %d.\n", getpid(), getppid(), getpgrp());
82         printf("\n- to check string is palindrome\n");
83
84         if (execlp("./palindrome.exe", "./palindrome.exe",
            "string.txt", NULL) == EXEC_ERROR)
85         {
86             printf("\nERROR: child 2 can not execute exec().\n");
87
88             exit(ERROR);
89         }
90
91         exit(OK);
92     }
93     else
94     {
95         child_pid = wait(&status);
96         printf("\nChild 2 END: PID = %d, status = %d\n",
            child_pid, status);
97
98         printf("\nParent: PID = %d, GPID = %d, child 2 PID =
            %d\n", getpid(), getpgrp(), child_pid1);
99         check_status(status);
100    }
101
102    return OK;
103 }

```

```
regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ ./main_03.exe
<<<<< Task 3 : Children do other programs with execlp() >>>>>

Child 1 START: PID = 56661, PPID = 56660, GPID = 56660.
- to sort array
Before sort: 3 9 6 2 0 6 3 -2 5 -1
After sort: -2 -1 0 2 3 3 5 6 6 9

Child 1 END: PID = 56661, status = 0.
Parent: PID = 56660, GPID = 56660, child 1 PID = 56661.
Child exited correctly with code 0.

Child 2 START: PID = 56662, PPID = 56660, GPID = 56660.
- to check string is palindrome
3 2 1 2 3 is palindrome

Child 2 END: PID = 56662, status = 0
Parent: PID = 56660, GPID = 56660, child 2 PID = 56661
Child exited correctly with code 0.
```

Рисунок 4 – Демонстрация работы программы

Задание 4

Процессы-потомки пишут разные сообщения в один неименованный программный канал, созданный системным вызовом `pipe()`. Процесс-предок считывает сообщения процессов-потомков и выводит сообщения на экран.

Листинг 4 – Системный вызов pipe()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6 #include <string.h>
7
8 #define FORK_ERROR -1
9 #define PIPE_ERROR -1
10 #define ERROR      1
11 #define OK          0
12
13
14 #define TASK          "\n<<<<< Task 4 : Messaging with pipe()
15     >>>>>\n\n"
16
17 #define READ  0
18 #define WRITE 1
19
20 #define MSG1  "Hello , child 2.\n"
21 #define MSG2  "How are you , child 1?\n"
22 #define LEN   50
23
24 void check_status(int status)
25 {
26     if (WIFEXITED(status))
27     {
28         printf("Child exited correctly with code %d.\n",
29             WEXITSTATUS(status));
30
31         return;
32     }
33     else if (WIFSIGNALED(status))
34     {
35         printf("Child exited with non-interceptable signal
36             %d.\n", WTERMSIG(status));
37
38         return;
39     }
40 }
```

```

38     else if (WIFSTOPPED(status))
39     {
40         printf("Child stopped with signal %d.\n",
41             WSTOPSIG(status));
42
43         return;
44     }
45 }
46
47 int main(void)
48 {
49     printf(TASK);
50
51     int fd[2];
52
53     pid_t child_pid1, child_pid2, child_pid;
54     int status;
55
56     char msgs[LEN] = {0};
57
58     if (pipe(fd) == PIPE_ERROR)
59     {
60         perror("\nCan't pipe.\n");
61         exit(ERROR);
62     }
63
64     if ((child_pid1 = fork()) == FORK_ERROR)
65     {
66         perror("\nCan't fork child 1.\n");
67         exit(ERROR);
68     }
69     else if (child_pid1 == 0)
70     {
71         printf("\nChild 1: PID = %d, PPID = %d, GPID = %d.\n",
72             getpid(), getppid(), getpgrp());
73
74         close(fd[READ]);
75         write(fd[WRITE], MSG1, strlen(MSG1));
76
77         exit(OK);
78     }

```

```

77
78
79     if ((child_pid2 = fork()) == FORK_ERROR)
80     {
81         perror("\nCan't fork child 2.\n");
82         exit(ERROR);
83     }
84     else if (child_pid2 == 0)
85     {
86         printf("Child 2: PID = %d, PPID = %d, GPID = %d.\n",
87             getpid(), getppid(), getpgrp());
88
89         close(fd[READ]);
90         write(fd[WRITE], MSG2, strlen(MSG2));
91
92         exit(OK);
93     }
94
95     child_pid = wait(&status);
96     printf("\n\nChild 1 has finished: PID = %d, status = %d.\n",
97         child_pid, status);
98     check_status(status);
99
100    child_pid = wait(&status);
101    printf("\n\nChild 2 has finished: PID = %d, status = %d.\n",
102        child_pid, status);
103    check_status(status);
104
105    close(fd[WRITE]);
106    read(fd[READ], msgs, LEN);
107    printf("\nChlds wrote : \n%s\n", msgs);
108
109    return OK;
110 }

```

```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ ./main_04.exe

<<<<< Task 4 : Messaging with pipe() >>>>>

Child 1: PID = 17699, PPID = 17698, GPID = 17698.
Child 2: PID = 17700, PPID = 17698, GPID = 17698.

Child 1 has fihished: PID = 17699, status = 0.
Child exited correctly with code 0.

Child 2 has fihished: PID = 17700, status = 0.
Child exited correctly with code 0.

Childs wrote :
Hello, child 2.
How are you, child 1?

```

Рисунок 5 – Демонстрация работы программы

Задание 5

Процесс-предок и процессы-потомки обмениваются сообщениями аналогично заданию 4. При помощи сигнала меняется ход выполнения программы: при получении сигнала от нажатия Ctrl + Z первый процесс-потомок записывает сообщение в канал, при получении сигнала от нажатия Ctrl + C второй процесс-потомок записывает сообщение в канал.

Листинг 5 – Системный вызов signal()

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <signal.h>
8
9 #define FORK_ERROR -1
10 #define PIPE_ERROR -1
11 #define ERROR      1
12 #define OK         0
13
14 #define PAUSE      5

```



```

15
16 #define TASK          "\n<<<<< Task 5 : Signal handler with
    signal() >>>>>\n\n"
17
18 #define READ  0
19 #define WRITE 1
20
21 #define MSG1  "Hello , child 2.\n"
22 #define MSG2  "How are you , child 1?\n"
23 #define LEN    50
24
25 #define TRUE    1
26 #define FALSE   0
27
28 #define SIGNAL_MSG "\nPress: CTRL + Z – for msgs from childs\n"
29
30
31 int flag = FALSE;
32
33
34 void check_status(int status)
35 {
36     if (WIFEXITED(status))
37     {
38         printf("Child exited correctly with code %d.\n",
39             WEXITSTATUS(status));
40
41         return;
42     }
43     else if (WIFSIGNALED(status))
44     {
45         printf("Child exited with non-interceptable signal
46             %d.\n", WTERMSIG(status));
47
48         return;
49     }
50     else if (WIFSTOPPED(status))
51     {
52         printf("Child stopped with signal %d.\n",
53             WSTOPSIG(status));

```

```

52         return;
53     }
54 }
55
56
57 void catch_ctrlz(int signal)
58 {
59     flag = TRUE;
60     printf("\nCaught signal = %d.\n", signal);
61 }
62
63
64 int main(void)
65 {
66     printf(TASK);
67     printf(SIGNAL_MSG);
68
69     signal(SIGTSTP, catch_ctrlz);
70     sleep(PAUSE);
71
72     int fd[2];
73
74     pid_t child_pid1, child_pid2, child_pid;
75     int status;
76
77     char msgs[LEN] = {0};
78
79     if (pipe(fd) == PIPE_ERROR)
80     {
81         perror("\nCan't pipe.\n");
82         exit(ERROR);
83     }
84
85
86     if ((child_pid1 = fork()) == FORK_ERROR)
87     {
88         perror("\nCan't fork child 1.\n");
89         exit(ERROR);
90     }
91     else if (child_pid1 == 0)
92     {

```

```

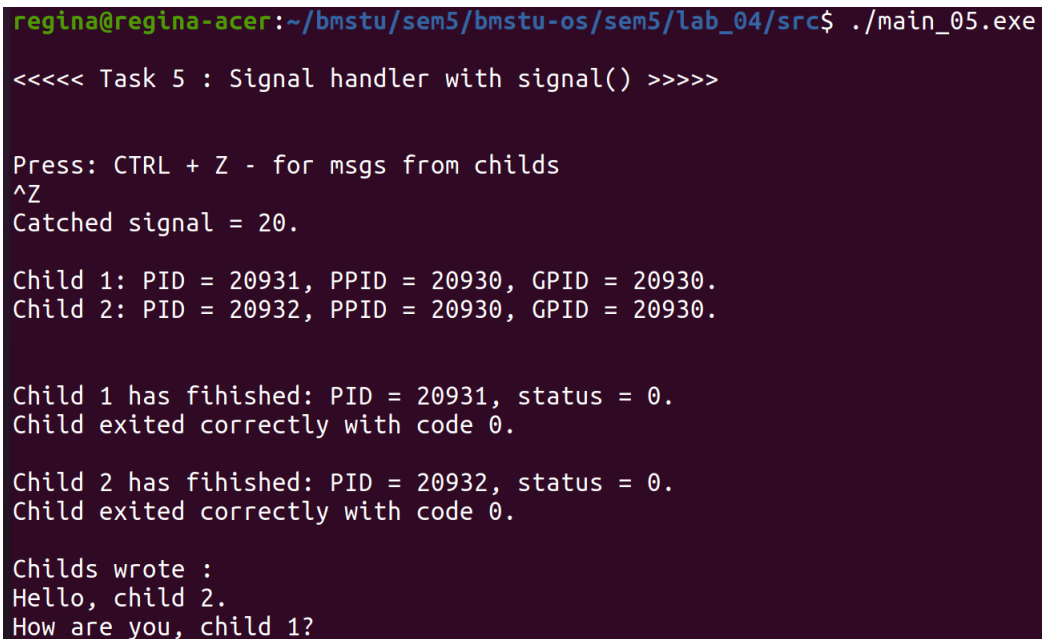
93     printf("\nChild 1: PID = %d, PPID = %d, GPID = %d.\n",
94           getpid(), getppid(), getpgrp());
95
96     if (flag == TRUE)
97     {
98         close(fd[READ]);
99         write(fd[WRITE], MSG1, strlen(MSG1));
100     }
101     exit(OK);
102 }
103
104
105 if ((child_pid2 = fork()) == FORK_ERROR)
106 {
107     perror("\nCan't fork child 2.\n");
108     exit(ERROR);
109 }
110 else if (child_pid2 == 0)
111 {
112     printf("Child 2: PID = %d, PPID = %d, GPID = %d.\n",
113           getpid(), getppid(), getpgrp());
114
115     if (flag == TRUE)
116     {
117         close(fd[READ]);
118         write(fd[WRITE], MSG2, strlen(MSG2));
119     }
120     exit(OK);
121 }
122
123 child_pid = wait(&status);
124 printf("\n\nChild 1 has fihished: PID = %d, status = %d.\n",
125       child_pid, status);
126 check_status(status);
127
128 child_pid = wait(&status);
129 printf("\n\nChild 2 has fihished: PID = %d, status = %d.\n",
130       child_pid, status);
131 check_status(status);

```

```

130
131     close(fd[WRITE]);
132     read(fd[READ], msgs, LEN);
133     printf("\nChlds wrote : \n%s\n", msgs);
134
135     return OK;
136 }

```



```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ ./main_05.exe

<<<<< Task 5 : Signal handler with signal() >>>>>

Press: CTRL + Z - for msgs from chlds
^Z
Caught signal = 20.

Child 1: PID = 20931, PPID = 20930, GPID = 20930.
Child 2: PID = 20932, PPID = 20930, GPID = 20930.

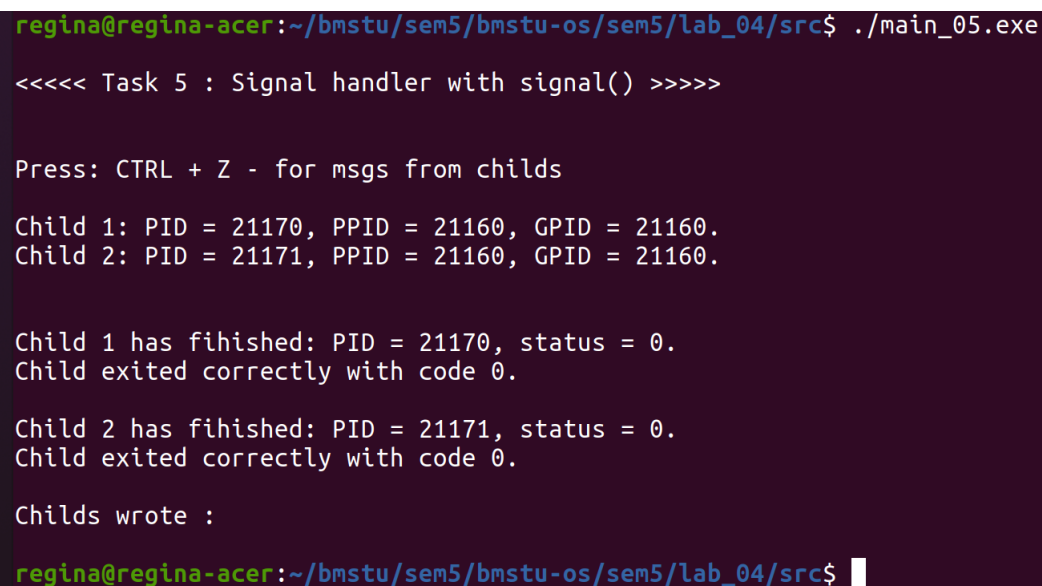
Child 1 has fihished: PID = 20931, status = 0.
Child exited correctly with code 0.

Child 2 has fihished: PID = 20932, status = 0.
Child exited correctly with code 0.

Chlds wrote :
Hello, child 2.
How are you, child 1?

```

Рисунок 6 – Демонстрация работы программы: сигнал от Ctrl + Z



```

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ ./main_05.exe

<<<<< Task 5 : Signal handler with signal() >>>>>

Press: CTRL + Z - for msgs from chlds

Child 1: PID = 21170, PPID = 21160, GPID = 21160.
Child 2: PID = 21171, PPID = 21160, GPID = 21160.

Child 1 has fihished: PID = 21170, status = 0.
Child exited correctly with code 0.

Child 2 has fihished: PID = 21171, status = 0.
Child exited correctly with code 0.

Chlds wrote :

regina@regina-acer:~/bmstu/sem5/bmstu-os/sem5/lab_04/src$ █

```

Рисунок 7 – Демонстрация работы программы: сигнал не вызван

Последовательность действий при вызове fork()

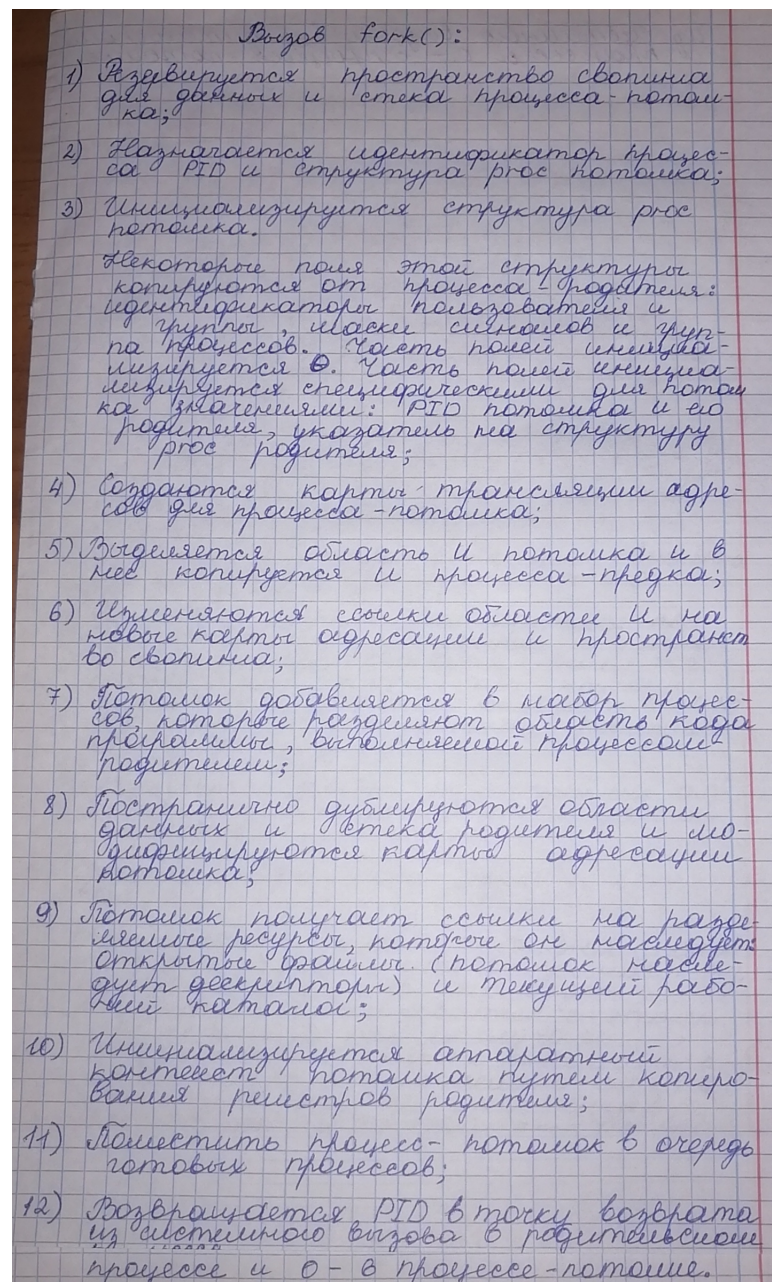


Рисунок 8 – Вызов fork()

Последовательность действий при вызове `exec()`

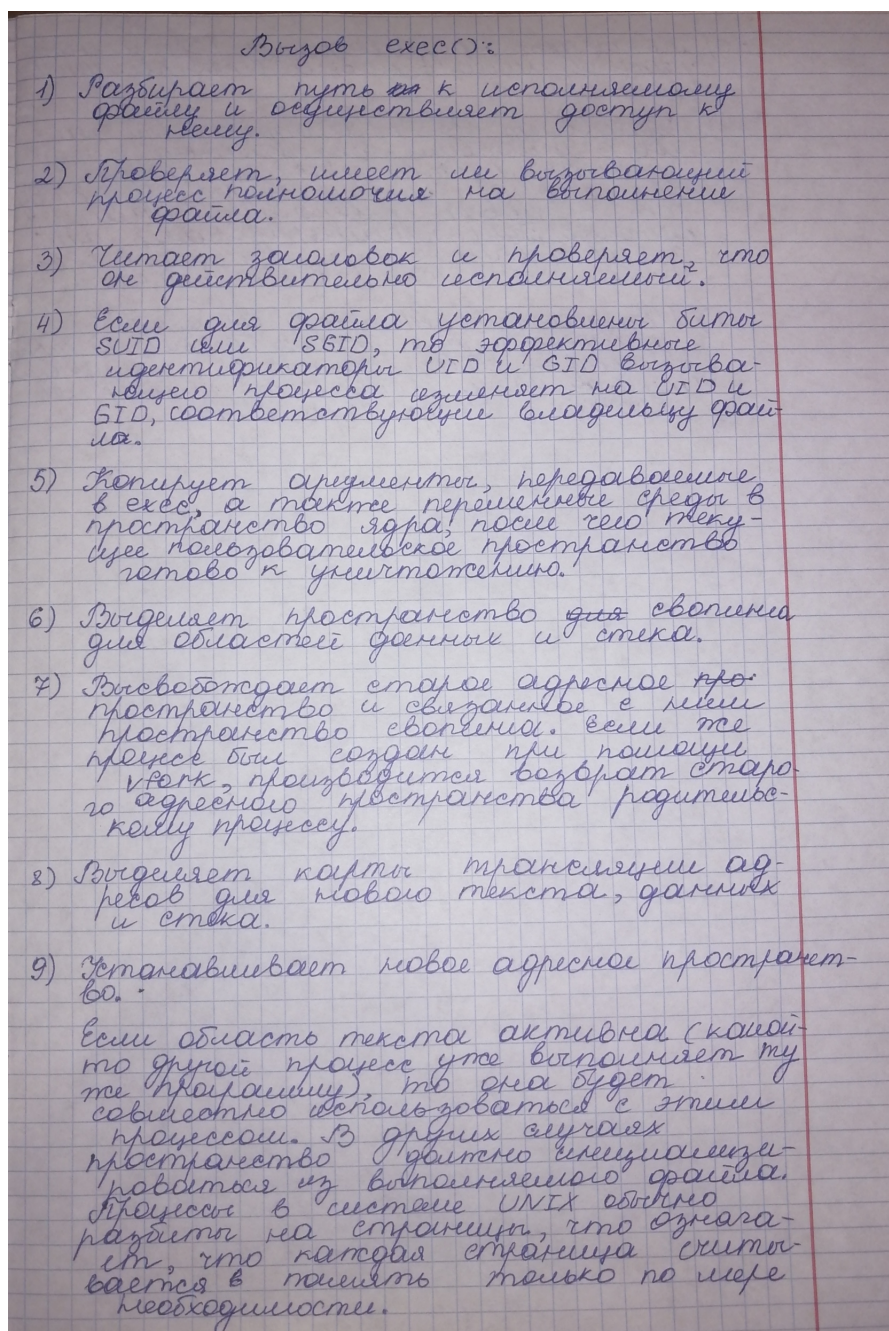


Рисунок 9 – Вызов `exec()` - 1

- 10) Копирует аргументы и переменные среды обратно в новую стек приоткрыв.
- 11) Сопоставляет все обработчики символов в действия, определенные по умолчанию, так как функции обработчиков символов не существуют в новой программе. Символы, которые были проинтерпретированы или закомментированы, перед вызовом ехес, остаются в тех же состояниях.
- 12) Инициализирует аппаратный контекст. При этом большинство регистров сопоставляется в 0, а указатель на начало получает значение точки входа программы.

Рисунок 10 – Вызов ехес() - 2