



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"**

Студент \_\_\_\_\_ Сукочева Алис  
Группа \_\_\_\_\_ ИУ7-53Б  
Название предприятия \_\_\_\_\_ МГТУ им. Н. Э. Баумана, каф. ИУ7  
Тема \_\_\_\_\_ Процессы. Системные вызовы `fork()` и `exec()`

Студент:	_____	Сукочева А.
	подпись, дата	Фамилия, И.О.
Преподаватель:	_____	Рязанова Н.Ю.
	подпись, дата	Фамилия, И. О.

Листинг 1 — Программа 1.

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5 #define OK 0
6 #define ERROR 1
7 #define SLEEP_TIME 2
8 #define ERROR_FORK -1
9
10 int main()
11 {
12     int childpid_1, childpid_2;
13
14     // Первый процесс.
15     // Создается дочерний процесс
16     if ((childpid_1 = fork()) == ERROR_FORK)
17     {
18         // Если при порождении процесса произошла ошибка.
19         perror("Can\'t fork.\n");
20         return ERROR;
21     }
22     else if (!childpid_1)
23     {
24         // Это процесс потомок.
25         printf("First child: id: %d ppid: %d pgrp: %d\n", getpid(),
26               getppid(), getpgrp());
27         sleep(SLEEP_TIME);
28         exit(OK);
29     }
30
31     // Аналогично 2 процесс.
32     if ((childpid_2 = fork()) == ERROR_FORK)
33     {
34         perror("Can\'t fork.\n");
35         return ERROR;
36     }
37     else if (!childpid_2)
38     {
39         // Это процесс потомок.
40         printf("Second child: id: %d ppid: %d pgrp: %d\n", getpid(),
41               getppid(), getpgrp());
42         sleep(SLEEP_TIME);
43         exit(OK);
44     }
45 }

```

```

44     printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
45           getpgrp(), childpid_1, childpid_2);
46     return OK;
47 }

```

```

lab_04 [master] ⚡ gcc f1.c
lab_04 [master] ⚡ ./a.out
Parent: id: 7102 pgrp: 7102 child1: 7103 child2: 7104
First child: id: 7103 ppid: 7102 pgrp: 7102
Second child: id: 7104 ppid: 7102 pgrp: 7102

```

Рисунок 0.1 — Результат работы программы 1.

```

1 S 1000 7103 5586 0 80 0 - 1126 hrttime pts/1 00:00:00 a.out
1 S 1000 7104 5586 0 80 0 - 1126 hrttime pts/1 00:00:00 a.out
4 R 1000 7116 5796 0 80 0 - 7533 - pts/2 00:00:00 ps

```

Рисунок 0.2 — Результат работы программы 1.

```

2009 5550 2009 2009 ? -1 St 0 0:00 gdm-session-worker [pam/gdm]
5547 5560 5560 5560 pts/0 5560 Ss+ 1000 0:00 zsh
1 5586 5586 5586 ? -1 Ss 1000 0:00 /lib/systemd/systemd --user
5586 5587 5586 5586 ? -1 S 1000 0:00 (sd-pam)

```

Рисунок 0.3 — Результат работы программы 1.

Листинг 2 — Программа 2.

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4 #include <stdlib.h>
5
6 #define OK 0
7 #define ERROR 1
8 #define ERROR_FORK -1
9 #define SLEEP_TIME 2
10
11 void check_status(int status);
12
13 int main()
14 {
15     int childpid_1, childpid_2;
16
17     if ((childpid_1 = fork()) == ERROR_FORK)
18     {
19         // Если при порождении процесса произошла ошибка.

```

```

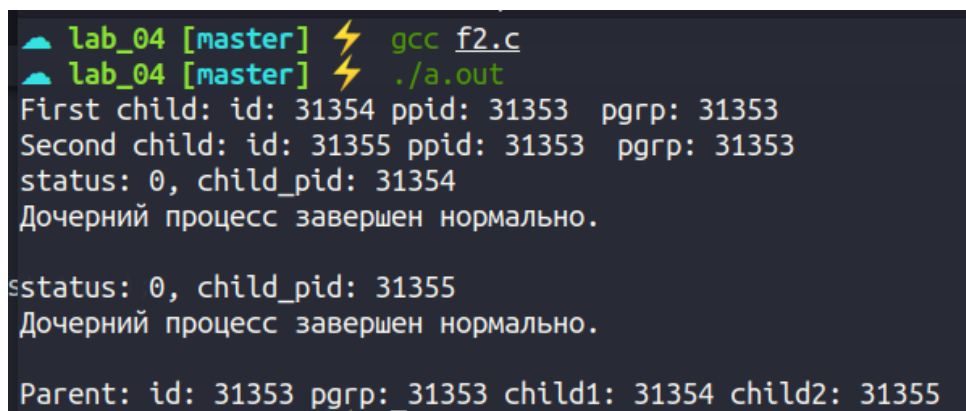
20     perror("Can\'t fork.\n");
21     return ERROR;
22 }
23 else if (!childpid_1)
24 {
25     // Это процесс потомок.
26     printf("First child: id: %d ppid: %d  pgrp: %d\n", getpid(),
27           getppid(), getpgrp());
28     sleep(SLEEP_TIME);
29     exit(OK);
30 }
31 // Аналогично 2 процесс.
32 if ((childpid_2 = fork()) == ERROR_FORK)
33 {
34     perror("Can\'t fork.\n");
35     return ERROR;
36 }
37 else if (!childpid_2)
38 {
39     // Это процесс потомок.
40     printf("Second child: id: %d ppid: %d  pgrp: %d\n", getpid(),
41           getppid(), getpgrp());
42     sleep(SLEEP_TIME);
43     exit(OK);
44 }
45 int status;
46 pid_t child_pid;
47
48 child_pid = wait(&status);
49 printf("status: %d, child_pid: %d\n", status, child_pid);
50 check_status(status);
51
52 child_pid = wait(&status);
53 printf("status: %d, child_pid: %d\n", status, child_pid);
54 check_status(status);
55
56 printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
57       getpgrp(), childpid_1, childpid_2);
58 return OK;
59 }
60
61 void check_status(int status)
62 {
63     if (WIFEXITED(status))

```

```

64     {
65         printf("Дочерний процесс завершен нормально.\n\n");
66         return;
67     }
68
69     if (WEXITSTATUS(status))
70     {
71         printf("Код завершения дочернего процесса %d.\n",
72             WIFEXITED(status));
73         return;
74     }
75
76     if (WIFSIGNALED(status))
77     {
78         printf("Дочерний процесс завершается перехватываемым сигналом
79             .\n");
80         printf("Номер сигнала %d.\n", WTERMSIG(status));
81         return;
82     }
83
84     if (WIFSTOPPED(status))
85     {
86         printf("Дочерний процесс остановился.\n");
87         printf("Номер сигнала %d.", WSTOPSIG(status));
88     }
89 }

```



```

lab_04 [master] gcc f2.c
lab_04 [master] ./a.out
First child: id: 31354 ppid: 31353  pggrp: 31353
Second child: id: 31355 ppid: 31353  pggrp: 31353
status: 0, child_pid: 31354
Дочерний процесс завершен нормально.

status: 0, child_pid: 31355
Дочерний процесс завершен нормально.

Parent: id: 31353 pggrp: 31353 child1: 31354 child2: 31355

```

Рисунок 0.4 — Результат работы программы 2.

### Листинг 3 — Программа 3.

```

1 #include <stdio.h>
2 #include <sys/wait.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5

```

```

6 #define OK 0
7 #define ERROR 1
8 #define ERROR_FORK -1
9 #define ERROR_EXEC -1
10
11 void check_status(int status);
12
13 int main()
14 {
15     int childpid_1, childpid_2;
16     int status;
17     pid_t child_pid;
18
19     if ((childpid_1 = fork()) == ERROR_FORK)
20     {
21         perror("Can\'t fork.\n");
22         return ERROR;
23     }
24     else if (!childpid_1) // Это процесс потомок (ребенок).
25     {
26         printf("First child: id: %d ppid: %d pgrp: %d\n", getpid(),
27             getppid(), getpgrp());
28
29         // р — определяет, что функция будет искать "дочернюю"
30         // программу в директориях, определяемых
31         // переменной среды DOS PATH. Без суффикса р поиск
32         // будет производиться только в рабочем каталоге.
33         if (execlp("cat", "cat", "text1.txt", NULL) == ERROR_EXEC)
34         {
35             perror("First child can\'t exec");
36             exit(ERROR);
37         }
38         exit(OK);
39     }
40
41     // Аналогично 2 процесс.
42     if ((childpid_2 = fork()) == ERROR_FORK)
43     {
44         perror("Can\'t fork.\n");
45         return ERROR;
46     }
47     else if (!childpid_2)
48     {
49         // Это процесс потомок.
50         printf("Second child: id: %d ppid: %d pgrp: %d\n", getpid(),
51             getppid(), getpgrp());
52         if (execlp("cat", "cat", "text2.txt", NULL) == ERROR_EXEC)

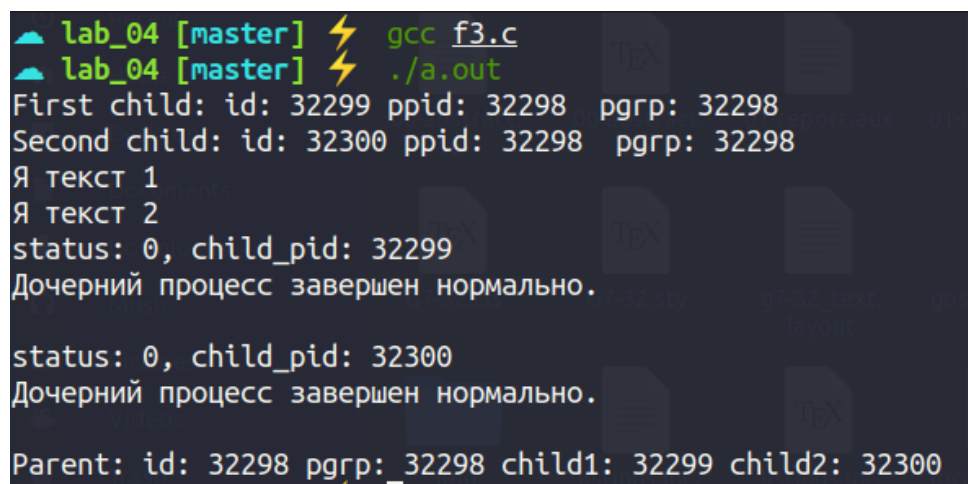
```

```

51     {
52         perror("Second child can\'t exec");
53         exit(ERROR);
54     }
55     exit(OK);
56 }
57
58 child_pid = wait(&status);
59 printf("status: %d, child_pid: %d\n", status, child_pid);
60 check_status(status);
61
62 child_pid = wait(&status);
63 printf("status: %d, child_pid: %d\n", status, child_pid);
64 check_status(status);
65
66 printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
        getpgrp(), childpid_1, childpid_2);
67
68 return OK;
69 }
70
71 void check_status(int status)
72 {
73     if (WIFEXITED(status))
74     {
75         printf("Дочерний процесс завершен нормально.\n\n");
76         return;
77     }
78
79     if (WEXITSTATUS(status))
80     {
81         printf("Код завершения дочернего процесса %d.\n",
                WIFEXITED(status));
82         return;
83     }
84
85     if (WIFSIGNALED(status))
86     {
87         printf("Дочерний процесс завершается неперехватываемым сигналом
                .\n");
88         printf("Номер сигнала %d.\n", WTERMSIG(status));
89         return;
90     }
91
92     if (WIFSTOPPED(status))
93     {
94         printf("Дочерний процесс остановился.\n");

```

```
95     printf("Номер сигнала %d.", WSTOPSIG(status));  
96 }  
97 }
```



```
lab_04 [master] gcc f3.c  
lab_04 [master] ./a.out  
First child: id: 32299 ppid: 32298 pgrp: 32298  
Second child: id: 32300 ppid: 32298 pgrp: 32298  
Я текст 1  
Я текст 2  
status: 0, child_pid: 32299  
Дочерний процесс завершен нормально.  
  
status: 0, child_pid: 32300  
Дочерний процесс завершен нормально.  
  
Parent: id: 32298 pgrp: 32298 child1: 32299 child2: 32300
```

Рисунок 0.5 — Результат работы программы 3.



## Листинг 4 — Программа 4.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/wait.h>
4 #include <unistd.h>
5 #include <string.h>
6
7 #define OK 0
8 #define ERROR 1
9 #define ERROR_FORK -1
10 #define ERROR_PIPE -1
11 #define LEN 32
12 #define FIRST_TEXT "First child write\n"
13 #define SECOND_TEXT "Second child write\n"
14
15 void check_status(int status);
16
17 int main()
18 {
19     int childpid_1, childpid_2;
20     int fd[2];
21
22     if (pipe(fd) == ERROR_PIPE)
23     {
24         perror("Can\'t pipe.\n");
25         return ERROR;
26     }
27
28     if ((childpid_1 = fork()) == ERROR_FORK)
29     {
30         // Если при порождении процесса произошла ошибка.
31         perror("Can\'t fork.\n");
32         return ERROR;
33     }
34     else if (!childpid_1) // Это процесс потомок.
35     {
36         close(fd[0]);
37         write(fd[1], FIRST_TEXT, strlen(FIRST_TEXT) + 1);
38         exit(OK);
39     }
40
41     // Аналогично 2 процесс.
42     if ((childpid_2 = fork()) == ERROR_FORK)
43     {
44         perror("Can\'t fork.\n");
45         return ERROR;

```

```

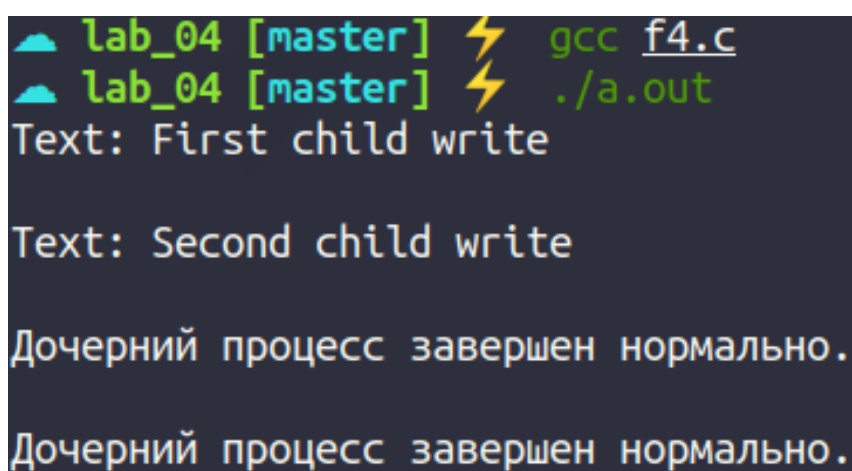
46     }
47     else if (!childpid_2) // Это процесс потомок.
48     {
49         close(fd[0]);
50         write(fd[1], SECOND_TEXT, strlen(SECOND_TEXT) + 1);
51         exit(OK);
52     }
53
54     char text[LEN], text2[LEN];
55     pid_t child_pid;
56     int status;
57
58     close(fd[1]);
59
60     read(fd[0], text, LEN);
61     read(fd[0], text2, LEN);
62
63     printf("Text: %s\n", text);
64     printf("Text: %s\n", text2);
65
66     child_pid = wait(&status);
67     check_status(status);
68
69     child_pid = wait(&status);
70     check_status(status);
71
72     return OK;
73 }
74
75 void check_status(int status)
76 {
77     if (WIFEXITED(status))
78     {
79         printf("Дочерний процесс завершен нормально.\n\n");
80         return;
81     }
82
83     if (WEXITSTATUS(status))
84     {
85         printf("Код завершения дочернего процесса %d.\n",
86                WIFEXITED(status));
87         return;
88     }
89
90     if (WIFSIGNALED(status))
91     {

```

```

91     printf("Дочерний процесс завершается перехватываемым сигналом
        .\n");
92     printf("Номер сигнала %d.\n", WTERMSIG(status));
93     return;
94 }
95
96 if (WIFSTOPPED(status))
97 {
98     printf("Дочерний процесс остановился.\n");
99     printf("Номер сигнала %d.", WSTOPSIG(status));
100 }
101 }

```



```

lab_04 [master] gcc f4.c
lab_04 [master] ./a.out
Text: First child write

Text: Second child write

Дочерний процесс завершен нормально.
Дочерний процесс завершен нормально.

```

Рисунок 0.6 — Результат работы программы 4.

Листинг 5 — Программа 5.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/wait.h>
4  #include <unistd.h>
5  #include <signal.h>
6  #include <stdbool.h> // bool.
7
8  #define OK 0
9  #define ERROR 1
10 #define ERROR_FORK -1
11 #define ERROR_PIPE -1
12 #define LEN 32
13
14 _Bool flag = false;
15
16 void check_status(int status);
17

```

```

18 void catch_sig(int sig_numb)
19 {
20     flag = true;
21     printf("catch_sig: %d\n", sig_numb);
22 }
23
24 int main()
25 {
26     signal(SIGINT, catch_sig);
27
28     int childpid_1, childpid_2;
29     int fd[2];
30
31     if (pipe(fd) == ERROR_PIPE)
32     {
33         perror("Can\'t pipe.\n");
34         return ERROR;
35     }
36
37     if ((childpid_1 = fork()) == ERROR_FORK)
38     {
39         // Если при порождении процесса произошла ошибка.
40         perror("Can\'t fork.\n");
41         return ERROR;
42     }
43     else if (!childpid_1) // Это процесс потомок.
44     {
45         close(fd[0]);
46         write(fd[1], "First child write\n", LEN);
47         exit(OK);
48     }
49
50     // Аналогично 2 процесс.
51     if ((childpid_2 = fork()) == ERROR_FORK)
52     {
53         perror("Can\'t fork.\n");
54         return ERROR;
55     }
56     else if (!childpid_2) // Это процесс потомок.
57     {
58         close(fd[0]);
59         write(fd[1], "Second child write\n", LEN);
60         exit(OK);
61     }
62
63     char text[LEN], text2[LEN];
64     pid_t child_pid;

```

```

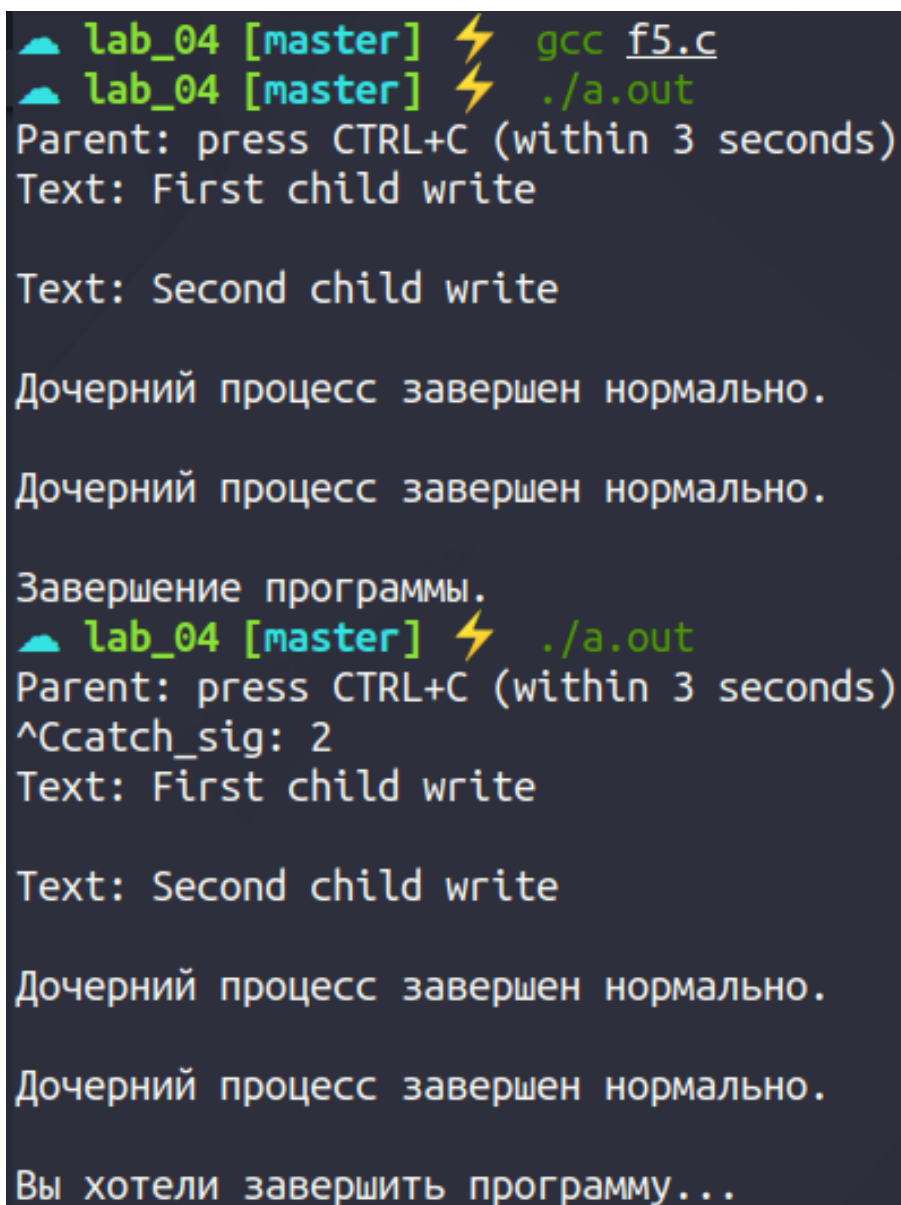
65     int status;
66
67     printf("Parent: press CTRL+C (within 3 seconds)\n");
68     sleep(3);
69
70     close(fd[1]);
71
72     read(fd[0], text, LEN);
73     read(fd[0], text2, LEN);
74
75     printf("Text: %s\n", text);
76     printf("Text: %s\n", text2);
77
78     child_pid = wait(&status);
79     check_status(status);
80
81     child_pid = wait(&status);
82     check_status(status);
83
84     if (flag)
85         printf("Вы хотели завершить программу...\n");
86     else
87         printf("Завершение программы.\n");
88
89     return OK;
90 }
91
92 void check_status(int status)
93 {
94     if (WIFEXITED(status))
95     {
96         printf("Дочерний процесс завершен нормально.\n\n");
97         return;
98     }
99
100     if (WEXITSTATUS(status))
101     {
102         printf("Код завершения дочернего процесса %d.\n",
103                WIFEXITED(status));
104         return;
105     }
106
107     if (WIFSIGNALED(status))
108     {
109         printf("Дочерний процесс завершается перехватываемым сигналом
110                .\n");
111         printf("Номер сигнала %d.\n", WTERMSIG(status));

```

```

110     return;
111 }
112
113 if (WIFSTOPPED(status))
114 {
115     printf("Дочерний процесс остановился.\n");
116     printf("Номер сигнала %d.", WSTOPSIG(status));
117 }
118 }

```



```

lab_04 [master] gcc f5.c
lab_04 [master] ./a.out
Parent: press CTRL+C (within 3 seconds)
Text: First child write

Text: Second child write

Дочерний процесс завершен нормально.
Дочерний процесс завершен нормально.

Завершение программы.
lab_04 [master] ./a.out
Parent: press CTRL+C (within 3 seconds)
^Ccatch_sig: 2
Text: First child write

Text: Second child write

Дочерний процесс завершен нормально.
Дочерний процесс завершен нормально.

Вы хотели завершить программу...

```

Рисунок 0.7 — Результат работы программы 5.