



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.

```

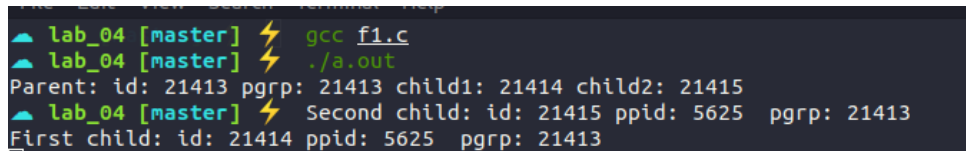
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5 #define OK 0
6 #define ERROR 1
7 #define SLEEP_TIME 2
8 #define ERROR_FORK -1
9
10 int main()
11 {
12     int childpid_1, childpid_2;
13
14     // Первый процесс.
15     // Создается дочерний процесс
16     if ((childpid_1 = fork()) == 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         sleep(SLEEP_TIME);
26         printf("First child: id: %d ppid: %d pgrp: %d\n", getpid(),
27             getppid(), getpgrp());
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         sleep(SLEEP_TIME);
41         printf("Second child: id: %d ppid: %d pgrp: %d\n", getpid(),
42             getppid(), getpgrp());
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 }

```

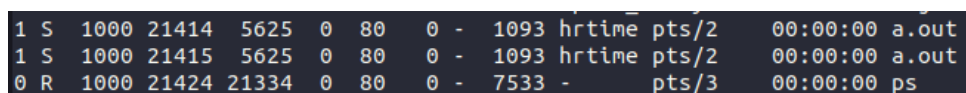


```

lab_04 [master] gcc f1.c
lab_04 [master] ./a.out
Parent: id: 21413 pgrp: 21413 child1: 21414 child2: 21415
lab_04 [master] Second child: id: 21415 ppid: 5625 pgrp: 21413
First child: id: 21414 ppid: 5625 pgrp: 21413

```

Рисунок 0.1 — Результат работы программы 1.

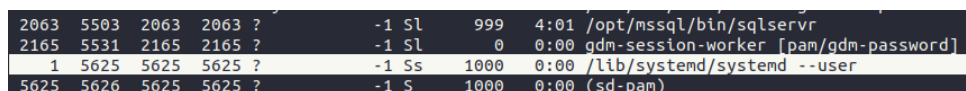


```

1 S 1000 21414 5625 0 80 0 - 1093 hrttime pts/2 00:00:00 a.out
1 S 1000 21415 5625 0 80 0 - 1093 hrttime pts/2 00:00:00 a.out
0 R 1000 21424 21334 0 80 0 - 7533 - pts/3 00:00:00 ps

```

Рисунок 0.2 — Новый parent pid у процессов-сирот.



```

2063 5503 2063 2063 ? -1 Sl 999 4:01 /opt/mssql/bin/sqlservr
2165 5531 2165 2165 ? -1 Sl 0 0:00 gdm-session-worker [pam/gdm-password]
1 5625 5625 5625 ? -1 Ss 1000 0:00 /lib/systemd/systemd --user
5625 5626 5625 5625 ? -1 S 1000 0:00 (sd-pam)

```

Рисунок 0.3 — Процесс, который усыновляет процессы-сироты.

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;

```

```

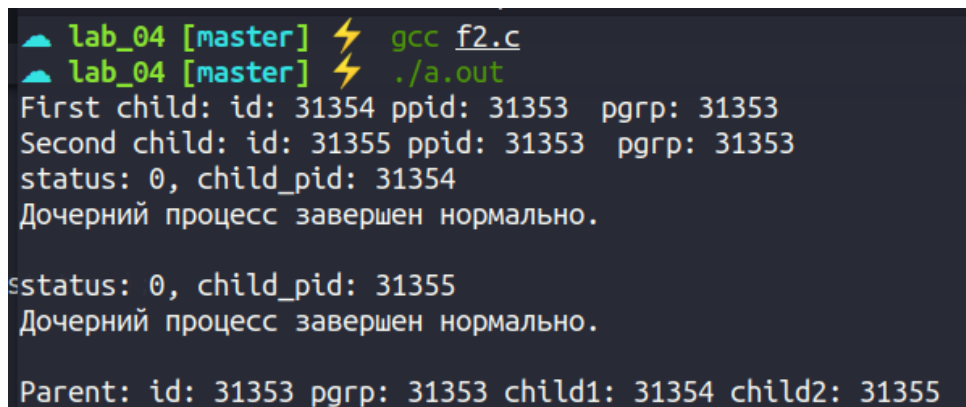
22     }
23     else if (!childpid_1)
24     {
25         // Это процесс потомок.
26         sleep(SLEEP_TIME);
27         printf("First child: id: %d ppid: %d  pgrp: %d\n", getpid(),
28             getppid(), getpgrp());
29         exit(OK);
30     }
31
32     // Аналогично 2 процесс.
33     if ((childpid_2 = fork()) == ERROR_FORK)
34     {
35         perror("Can\'t fork.\n");
36         return ERROR;
37     }
38     else if (!childpid_2)
39     {
40         // Это процесс потомок.
41         sleep(SLEEP_TIME);
42         printf("Second child: id: %d ppid: %d  pgrp: %d\n", getpid(),
43             getppid(), getpgrp());
44         exit(OK);
45     }
46
47     int status;
48     pid_t child_pid;
49
50     child_pid = wait(&status);
51     printf("status: %d, child_pid: %d\n", status, child_pid);
52     check_status(status);
53
54     child_pid = wait(&status);
55     printf("status: %d, child_pid: %d\n", status, child_pid);
56     check_status(status);
57
58     printf("Parent: id: %d pgrp: %d child1: %d child2: %d\n", getpid(),
59         getpgrp(), childpid_1, childpid_2);
60
61     return OK;
62 }
63
64 void check_status(int status)
65 {
66     if (WIFEXITED(status))
67     {
68         printf("Дочерний процесс завершен нормально.\n\n");
69     }
70 }

```

```

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 pgid: 31353
Second child: id: 31355 ppid: 31353 pgid: 31353
status: 0, child_pid: 31354
Дочерний процесс завершен нормально.
status: 0, child_pid: 31355
Дочерний процесс завершен нормально.
Parent: id: 31353 pgid: 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("echo", "echo", "Hello", "world!\n", NULL) ==
53             ERROR_EXEC)
54         {

```

```

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 перехватываемым 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: 11181 ppid: 11180 pgrp: 11180  
Second child: id: 11182 ppid: 11180 pgrp: 11180  
  
Я текст 1  
Hello world!  
  
Дочерний процесс завершен нормально.  
  
Дочерний процесс завершен нормально.  
  
Parent: id: 11180 pgrp: 11180 child1: 11181 child2: 11182
```

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;

```

```

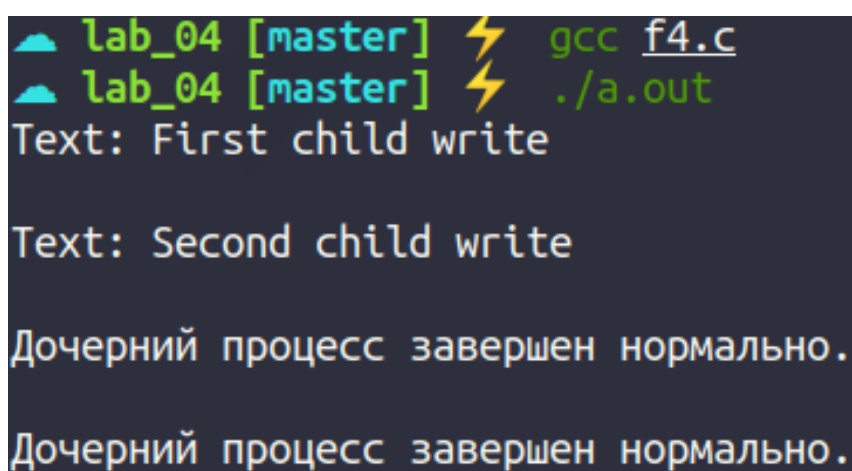
46     }
47     else if (!childpid_2) // Это процесс потомок.
48     {
49         close(fd[0]);
50         write(fd[1], SECOND_TEXT, strlen(SECOND_TEXT) + 1);
51         exit(OK);
52     }
53
54     char text[LEN], text2[LEN];
55     pid_t child_pid;
56     int status;
57
58     close(fd[1]);
59
60     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  #include <string.h>
8
9  #define OK 0
10 #define ERROR 1
11 #define ERROR_FORK -1
12 #define ERROR_PIPE -1
13 #define LEN 32
14 #define FIRST_TEXT "First child write\n"
15 #define SECOND_TEXT "Second child write\n"
16
17 _Bool flag = false;

```

```

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

```

```

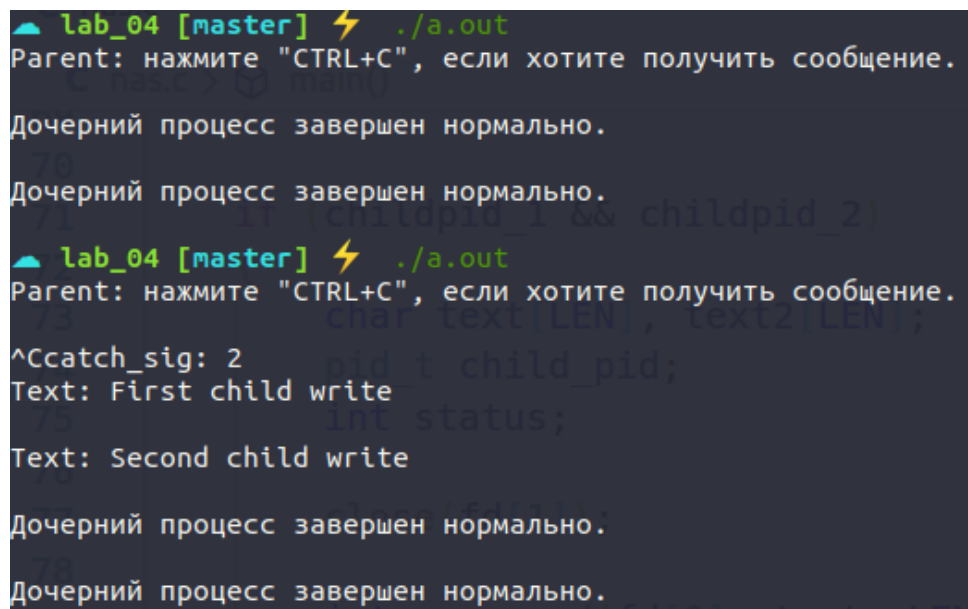
64         write(fd[1], SECOND_TEXT, strlen(SECOND_TEXT) + 1);
65         exit(OK);
66     }
67
68     if (childpid_1 && childpid_2)
69     {
70         char text[LEN], text2[LEN];
71         pid_t child_pid;
72         int status;
73
74         close(fd[1]);
75
76         int a = read(fd[0], text, LEN);
77
78         if (a)
79         {
80             read(fd[0], text2, LEN);
81
82             printf("Text: %s\n", text);
83             printf("Text: %s\n", text2);
84         }
85
86         child_pid = wait(&status);
87         check_status(status);
88
89         child_pid = wait(&status);
90         check_status(status);
91     }
92
93     return OK;
94 }
95
96 void check_status(int status)
97 {
98     if (WIFEXITED(status))
99     {
100         printf("Дочерний процесс завершен нормально.\n\n");
101         return;
102     }
103
104     if (WEXITSTATUS(status))
105     {
106         printf("Код завершения дочернего процесса %d.\n",
107                WIFEXITED(status));
108         return;
109     }

```

```

110     if (WIFSIGNALED(status))
111     {
112         printf("Дочерний процесс завершается неперехватываемым сигналом\n");
113         printf("Номер сигнала %d.\n", WTERMSIG(status));
114         return;
115     }
116
117     if (WIFSTOPPED(status))
118     {
119         printf("Дочерний процесс остановился.\n");
120         printf("Номер сигнала %d.", WSTOPSIG(status));
121     }
122 }

```



```

lab_04 [master] ./a.out
Parent: нажмите "CTRL+C", если хотите получить сообщение.
Дочерний процесс завершен нормально.
Дочерний процесс завершен нормально.
lab_04 [master] ./a.out
Parent: нажмите "CTRL+C", если хотите получить сообщение.
^Ccatch_sig: 2
Text: First child write
Text: Second child write
Дочерний процесс завершен нормально.
Дочерний процесс завершен нормально.

```

Рисунок 0.7 — Результат работы программы 5.