



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»

# Отчёт

## по лабораторной работе № 4

**Название:** 5 системных вызовов ОС UNIX/LINUX

**Дисциплина:** Операционные системы

Студент

ИУ7-55Б

(Группа)

Д.В. Сусликов

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Н.Ю. Рязанова

(Подпись, дата)

(И.О. Фамилия)

*Москва, 2020*

## **Содержание**

<b>Задание 1</b>	<b>3</b>
<b>Задание 2</b>	<b>6</b>
<b>Задание 3</b>	<b>10</b>
<b>Задание 4</b>	<b>13</b>
<b>Задание 5</b>	<b>16</b>

## Задание 1

Написать программу, запускающую не менее двух новых процессов системным вызовом `fork()`. В предке вывести собственный идентификатор (функция `getpid()`), идентификатор группы ( функция `getpgrp()`) и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка (функция `getppid()`) и идентификатор группы. Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника.

Программа представлена в Листинге 1.

Листинг 1 - Программа 1.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  void print_child_info(int child_num, char *status)
6  {
7      printf("\n%s\n\
8      child:\n\
9      number = %d\n\
10     pid = %d\n\
11     parent = %d\n\
12     group = %d\n", status, child_num, getpid(), \
13     getppid(), getpgrp());
14 }
15
16 pid_t fork_child(int child_num)
17 {
18     pid_t child = fork();
19     if (child == -1)
20     {
21         perror("fork error");
```

```

22     exit(1);
23 }
24 else if (child == 0)
25 {
26     print_child_info(child_num, "S T A R T");
27     sleep(3);
28     print_child_info(child_num, "E N D I N G");
29     exit(0);
30 }
31 return child;
32 }
33
34 int main()
35 {
36     pid_t child_1 = fork_child(1);
37     pid_t child_2 = fork_child(2);
38     pid_t child_3 = fork_child(3);
39
40     printf("\nparent:\n\
41     pid = %d\n\
42     group = %d\n\
43     child_1 = %d\n\
44     child_2 = %d\n\
45     child_3 = %d\n", getpid(), getpgrp(), child_1, child_2,
        child_3);
46
47     return 0;
48 }

```

Результат работы программы показан на Рисунке 1.

```
S T A R T
  child:
    number = 3
    pid = 5239
    parent = 5236
    group = 5236

S T A R T
  child:
    number = 2
    pid = 5238
    parent = 5236
    group = 5236

S T A R T
  child:
    number = 1
    pid = 5237
    parent = 5236
    group = 5236

parent:
  pid = 5236
  group = 5236
  child_1 = 5237
  child_2 = 5238
  child_3 = 5239
daniil@daniil-VirtualBox:~/Рабочий стол/OS$
E N D I N G
  child:
    number = 3
    pid = 5239
    parent = 1360
    group = 5236

E N D I N G
  child:
    number = 2
    pid = 5238
    parent = 1360
    group = 5236

E N D I N G
  child:
    number = 1
    pid = 5237
    parent = 1360
    group = 5236
```

**Рисунок 1** – Результат работы программы 1

## Задание 2

Написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов `wait()`. Убедиться, что в этом случае идентификатор процесса потомка на 1 больше идентификатора процесса-предка.

Программа показана в Листинге 2.

Листинг 2 - Программа 2.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  void print_child_info(int child_num, char *status)
8  {
9      printf("\n%s\n\
10     child:\n\
11     number = %d\n\
12     pid = %d\n\
13     parent = %d\n\
14     group = %d\n", status, child_num, getpid(), getppid(), getpgrp
        ());
15 }
16
17 pid_t fork_child(int child_num)
18 {
19     pid_t child = fork();
20     if (child == -1)
21     {
22         perror("fork error");
23         exit(1);
24     }
25     else if (child == 0)
26     {
```

```

27     print_child_info(child_num, "S T A R T");
28     sleep(3);
29     print_child_info(child_num, "E N D I N G");
30     exit(0);
31 }
32 return child;
33 }
34
35 void wait_for_child()
36 {
37     int stat_val;
38     pid_t child = wait(&stat_val);
39     printf("Child has finished: PID = %d\n", child);
40     if (WIFEXITED(stat_val))
41         printf("Child = %d ended normal with code %d\n", child,
42             WEXITSTATUS(stat_val));
43     else if (WIFSIGNALED(stat_val))
44         printf("Child = %d ended with a non-intercepted signal with
45             code %d\n", child, WTERMSIG(stat_val));
46     else if (WIFSTOPPED(stat_val))
47         printf("Child = %d stopped with code %d\n", child, WSTOPSIG(
48             stat_val));
49 }
50
51 int main()
52 {
53     pid_t child_1 = fork_child(1);
54     pid_t child_2 = fork_child(2);
55     pid_t child_3 = fork_child(3);
56     printf("\nparent:\n\
57     pid = %d\n\
58     group = %d\n\
59     child_1 = %d\n\
60     child_2 = %d\n\
61     child_3 = %d\n", getpid(), getpgrp(), child_1, child_2,

```

```
        child_3);  
59  
60    wait_for_child();  
61    wait_for_child();  
62    wait_for_child();  
63  
64    return 0;  
65 }
```



Результат работы программы изображен на Рисунке 2.

```
parent:
  pid = 5770
  group = 5770
  child_1 = 5771
  child_2 = 5772
  child_3 = 5773

S T A R T
  child:
    number = 3
    pid = 5773
    parent = 5770
    group = 5770

S T A R T
  child:
    number = 2
    pid = 5772
    parent = 5770
    group = 5770

S T A R T
  child:
    number = 1
    pid = 5771
    parent = 5770
    group = 5770

E N D I N G
  child:
    number = 3
    pid = 5773
    parent = 5770
    group = 5770
Child has finished: PID = 5773
Child = 5773 ended normal with code 0

E N D I N G
  child:
    number = 2
    pid = 5772
    parent = 5770
    group = 5770
Child has finished: PID = 5772
Child = 5772 ended normal with code 0

E N D I N G
  child:
    number = 1
    pid = 5771
    parent = 5770
    group = 5770
Child has finished: PID = 5771
Child = 5771 ended normal with code 0
```

**Рисунок 2** – Результат работы программы 2

### Задание 3

Написать программу, в которой процесс-потомок вызывает системный вызов `exec()`, а процесс-предок ждет завершения процесса-потомка. Следует создать не менее двух потомков.

Программа представлена в Листинге 3.

Листинг 3 - Программа 3.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  void print_child_info(int child_num)
8  {
9      printf("\nchild:\n\
10     number = %d\n\
11     pid = %d\n\
12     parent = %d\n\
13     group = %d\n", child_num, getpid(), getppid(), getpgrp());
14 }
15
16 pid_t fork_child(int child_num, char *path, char *arg0)
17 {
18     pid_t child = fork();
19     if (child == -1)
20     {
21         perror("fork error");
22         exit(1);
23     }
24     else if (child == 0)
25     {
26         print_child_info(child_num);
27         if (execl(path, arg0, NULL) == -1)
```

```

28     {
29         perror("exec error");
30         exit(1);
31     }
32
33 }
34 return child;
35 }
36
37 void wait_for_child()
38 {
39     int stat_val;
40     pid_t child = wait(&stat_val);
41     printf("Child has finished: PID = %d\n", child);
42     if (WIFEXITED(stat_val))
43         printf("Child %d ended normal with code %d\n", child,
44             WEXITSTATUS(stat_val));
45     else if (WIFSIGNALED(stat_val))
46         printf("Child %d ended with a non-intercepted signal with
47             code %d\n", child, WTERMSIG(stat_val));
48     else if (WIFSTOPPED(stat_val))
49         printf("Child %d stopped with code %d\n", child, WSTOPSIG(
50             stat_val));
51 }
52
53 int main()
54 {
55     pid_t child_1 = fork_child(1, "/bin/ls", "ls");
56     pid_t child_2 = fork_child(2, "/bin/ps", "ps");
57
58     printf("\nparent:\n\
59     pid = %d\n\
60     group = %d\n\
61     child_1 = %d\n\
62     child_2 = %d\n", getpid(), getpgrp(), child_1, child_2);

```

```

60
61     wait_for_child();
62     wait_for_child();
63
64     return 0;
65 }

```

Результат работы программы показан на Рисунке 3.

```

parent:
  pid = 5993
  group = 5993
  child_1 = 5994
  child_2 = 5995

child:
  number = 2
  pid = 5995
  parent = 5993
  group = 5993

child:
  number = 1
  pid = 5994
  parent = 5993
  group = 5993
3.out task1.c task2.c task3.c task4.c task5.c
Child has finished: PID = 5994
Child 5994 ended normal with code 0
  PID TTY          TIME CMD
  4888 pts/0        00:00:00 bash
  5993 pts/0        00:00:00 3.out
  5995 pts/0        00:00:00 ps
Child has finished: PID = 5995
Child 5995 ended normal with code 0

```

**Рисунок 3** – Результат работы программы 3

## Задание 4

Написать программу, в которой предок и потомок обмениваются сообщением через программный канал.

Программа показана в Листинге 4.

Листинг 4 - Программа 4.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  pid_t fork_child(int child_num, int *fd)
8  {
9      pid_t child = fork();
10     if (child == -1)
11     {
12         perror("fork error");
13         exit(1);
14     }
15     else if (child == 0)
16     {
17         int pid_int = getpid();
18         void* pid = &pid_int;
19         close(fd[0]);
20         write(fd[1], pid, sizeof(pid));
21         exit(0);
22     }
23     return child;
24 }
25
26 void wait_for_child(int *fd)
27 {
28     int stat_val;
```

```

29     void* pid = malloc(sizeof(void*));
30
31     close(fd[1]);
32     read(fd[0], pid, sizeof(pid));
33
34     pid_t child = wait(&stat_val);
35     printf("Child %d send %d\n", child, *(int*)pid);
36     if (WIFEXITED(stat_val))
37         printf("Child %d completed normally with code %d\n", child,
38             WEXITSTATUS(stat_val));
39     else if (WIFSIGNALED(stat_val))
40         printf("Child %d ended with a non-intercepted signal with code
41             %d\n", child, WTERMSIG(stat_val));
42     else if (WIFSTOPPED(stat_val))
43         printf("Child %d stopped with code %d\n", child, WSTOPSIG(
44             stat_val));
45
46     free(pid);
47 }
48
49 int main()
50 {
51     int fd[2];
52     if (pipe(fd) == -1)
53     {
54         perror("pipe error");
55         exit(1);
56     }
57
58     pid_t child_1 = fork_child(1, fd);
59     pid_t child_2 = fork_child(2, fd);
60
61     printf("\nparent:\n\
62     pid = %d\n\
63     group = %d\n\

```

```

61     child_1 = %d\n\
62     child_2 = %d\n", getpid(), getpgrp(), child_1, child_2);
63
64     wait_for_child(fd);
65     wait_for_child(fd);
66
67     return 0;
68 }

```

Результат работы программы изображен на Рисунке 4.

```

parent:
  pid = 6182
  group = 6182
  child_1 = 6183
  child_2 = 6184
Child 6184 send 6184
Child 6184 completed normally with code 0
Child 6183 send 6183
Child 6183 completed normally with code 0

```

**Рисунок 4** – Результат работы программы 4

## Задание 5

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

Программа представлена в Листинге 5.

Листинг 5 - Программа 5.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6  #include <string.h>
7
8  int flag = 0;
9
10 void catch_signal(int sig_num)
11 {
12     printf("\nCatch signal Cntrl + Z\n");
13     flag = 1;
14 }
15
16 void waiting_for_signal()
17 {
18     printf("Press Ctrl + Z to continue\n");
19     signal(SIGTSTP, catch_signal);
20     sleep(3);
21 }
22
23 pid_t fork_child(int child_num, int *fd)
24 {
25     pid_t child = fork();
26     if (child == -1)
27     {
28         perror("fork error");
```



```

29     exit(1);
30 }
31 else if (child == 0)
32 {
33     if (flag)
34     {
35         int pid_int = getpid();
36         void* pid = &pid_int;
37         close(fd[0]);
38         write(fd[1], pid, sizeof(pid));
39     }
40     exit(0);
41 }
42 return child;
43 }
44
45 void wait_for_child(int *fd)
46 {
47     int stat_val;
48     void *pid = (void*)malloc(sizeof(void*));
49     pid_t child = wait(&stat_val);
50     if (flag)
51     {
52         close(fd[1]);
53         read(fd[0], pid, sizeof(pid));
54         printf("Child %d send %d\n", child, *(int*)pid);
55     }
56     else
57         printf("Child %d send NONE\n", child);
58
59     if (WIFEXITED(stat_val))
60         printf("Child %d completed normally with code %d\n", child,
61             WEXITSTATUS(stat_val));
62     else if (WIFSIGNALED(stat_val))
63         printf("Child %d ended with a non-intercepted signal with code

```

```

        %d\n", child, WTERMSIG(stat_val));
63     else if (WIFSTOPPED(stat_val))
64         printf("Child %d stopped with code %d\n", child, WSTOPSIG(
            stat_val));
65
66     free(pid);
67 }
68
69 int main()
70 {
71     int fd[2];
72
73     if (pipe(fd) == -1)
74     {
75         perror("pipe error");
76         exit(1);
77     }
78
79     waiting_for_signal();
80
81     pid_t child_1 = fork_child(1, fd);
82     pid_t child_2 = fork_child(2, fd);
83
84     printf("\nparent:\n\
85     pid = %d\n\
86     group = %d\n\
87     child_1 = %d\n\
88     child_2 = %d\n", getpid(), getpgrp(), child_1, child_2);
89
90     wait_for_child(fd);
91     wait_for_child(fd);
92
93     return 0;
94 }

```

Результат работы программы при перехвате сигнала показан на Рисунке 5.

```
Press Ctrl + Z to continue
^Z
Catch signal Cntrl + Z

parent:
  pid = 6478
  group = 6478
  child_1 = 6479
  child_2 = 6480
Child 6480 send 6480
Child 6480 completed normally with code 0
Child 6479 send 6479
Child 6479 completed normally with code 0
```

**Рисунок 5** – Результат работы программы 5 при перехвате сигнала

Результат работы программы при отсутствии сигнала изображен на Рисунке 6.

```
Press Ctrl + Z to continue

parent:
  pid = 6481
  group = 6481
  child_1 = 6482
  child_2 = 6483
Child 6483 send NONE
Child 6483 completed normally with code 0
Child 6482 send NONE
Child 6482 completed normally with code 0
```

**Рисунок 6** – Результат работы программы 6 при отсутствии сигнала