



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

Студент \_\_\_\_\_ Топорков Павел \_\_\_\_\_

Группа \_\_\_\_\_ ИУ7-53Б \_\_\_\_\_

Дисциплина \_\_\_\_\_ Операционные системы \_\_\_\_\_

Преподаватель: \_\_\_\_\_ Рязанова Н. Ю. \_\_\_\_\_

подпись, дата

Фамилия, И.О.

Оценка \_\_\_\_\_

Москва — 2020 г.

# Оглавление

<b>1</b>	<b>Задание №1</b>	<b>2</b>
1.1	Задание . . . . .	2
1.2	Код программы . . . . .	2
1.3	Демонстрация работы программы . . . . .	3
<b>2</b>	<b>Задание №2</b>	<b>4</b>
2.1	Задание . . . . .	4
2.2	Код программы . . . . .	4
2.3	Демонстрация работы программы . . . . .	6
<b>3</b>	<b>Задание №3</b>	<b>7</b>
3.1	Задание . . . . .	7
3.2	Код программы . . . . .	7
3.3	Демонстрация работы программы . . . . .	9
<b>4</b>	<b>Задание №4</b>	<b>10</b>
4.1	Задание . . . . .	10
4.2	Код программы . . . . .	10
4.3	Демонстрация работы программы . . . . .	12
<b>5</b>	<b>Задание №5</b>	<b>13</b>
5.1	Задание . . . . .	13
5.2	Код программы . . . . .	13
5.3	Демонстрация работы программы . . . . .	16

# 1 Задание №1

## 1.1 Задание

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.

## 1.2 Код программы

Листинг 1.1: task1

```
0 #include <stdio.h>
1 #include <sys/types.h>
2 #include <unistd.h>
3
4 #define FORK_ERROR 1
5 #define OK 0
6
7
8 int main() {
9     pid_t first_ch, second_ch;
10
11     first_ch = fork();
12
13     if (first_ch == - 1) {
14         perror("Can't fork.\n ");
15         return FORK_ERROR;
16     } else if (first_ch == 0) {
17         sleep(2);
18         fprintf(stdout, "\n1st child process:\tpid = %d\tppid = %d\tgroup
id = %d\n",
19                 getpid(), getppid(), getpgrp());
20
21         return OK;
```

```

22     } else {
23         second_ch = fork();
24
25         if (second_ch == - 1) {
26             perror("Can't fork.\n ");
27             return FORK_ERROR;
28         } else if (second_ch == 0) {
29             sleep(2);
30             fprintf(stdout, "\n2nd child process:\tpid = %d\tppid = %d\tgroup id = %d\n",
31                 getpid(), getppid(), getpgrp());
32
33             return OK;
34         }
35
36         fprintf(stdout, "Parent process:\tpid = %d\tchild proc. id = %d, %d\tgroup id = %d\n",
37             getpid(), first_ch , second_ch, getpgrp());
38
39         return OK;
40     }
41
42     return OK;
43 }

```

### 1.3 Демонстрация работы программы

```

[ptrk@MacBook-Pro-Pavel] - [~/BMSTU/BMSTU-OS/lab04] - [728]
[17:15:43]
Parent process: pid = 52321      child proc. id = 52322, 52323      group id = 52321
1st child process:      pid = 52322      ppid = 1      group id = 52321
2nd child process:      pid = 52323      ppid = 1      group id = 52321

```

Рисунок 1.1

## 2 Задание №2

### 2.1 Задание

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.

### 2.2 Код программы

Листинг 2.1: task2

```
0 #include <stdio.h>
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4
5 #define FORK_ERROR 1
6 #define OK 0
7
8
9 int main() {
10     pid_t first_ch, second_ch;
11     int status, data;
12
13     first_ch = fork();
14
15     if (first_ch == - 1) {
16         perror("Can't fork.\n ");
17         return FORK_ERROR;
18     } else if (first_ch == 0) {
19         sleep(2);
20         fprintf(stdout, "\n1st child process:\tppid = %d\tppid = %d\tgroup
id = %d\n",
21                 getpid(), getppid(), getpgrp());
22
23         return OK;
24     } else {
25         data = wait(&status);
26
27         if (WIFEXITED(status)) {
```

```

28         fprintf(stdout, "Child process (%d) finished with code %d\n",
29                 data, WEXITSTATUS(status));
30     } else if (WIFSIGNALED(status)) {
31         fprintf(stdout, "Child process (%d) finished from signal with
code %d\n",
32                 data, WTERMSIG(status));
33     } else if (WIFSTOPPED(status)) {
34         fprintf(stdout, "Child process (%d) finished from signal with
code %d\n",
35                 data, WSTOPSIG(status));
36     }
37
38     second_ch = fork();
39
40     if (second_ch == - 1) {
41         perror("Can't fork.\n ");
42         return FORK_ERROR;
43     } else if (second_ch == 0) {
44         sleep(2);
45         fprintf(stdout, "\n2nd child process:\tpid = %d\tppid = %d\
tgroup id = %d\n",
46                 getpid(), getppid(), getpgrp());
47
48         return OK;
49     }
50
51     data = wait(&status);
52
53     if (WIFEXITED(status)) {
54         fprintf(stdout, "Child process (%d) finished with code %d\n",
55                 data, WEXITSTATUS(status));
56     } else if (WIFSIGNALED(status)) {
57         fprintf(stdout, "Child process (%d) finished from signal with
code %d\n",
58                 data, WTERMSIG(status));
59     } else if (WIFSTOPPED(status)) {
60         fprintf(stdout, "Child process %d finished from signal with
code %d\n",
61                 data, WSTOPSIG(status));
62     }
63
64     fprintf(stdout, "\nParent process:\tpid = %d\tchild proc. id = %d,
%d\tgroup id = %d\n",
65             getpid(), first_ch , second_ch, getpgrp());
66
67     return OK;
68 }
69

```

```
70     return OK;
71 }
```

## 2.3 Демонстрация работы программы

```
1st child process:      pid = 52972      ppid = 52971      group id = 52971
Child process (52972) finished with code 0

2nd child process:      pid = 52973      ppid = 52971      group id = 52971
Child process (52973) finished with code 0

Parent process: pid = 52971      child proc. id = 52972, 52973      group id = 52971
```

Рисунок 2.1

## 3 Задание №3

### 3.1 Задание

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

С помощью системного вызова `exec()` я запускаю в своей программе программы `date` и `ps` (`ps -al`).

### 3.2 Код программы

Листинг 3.1: task3

```
0 #include <stdio.h>
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4
5 #define FORK_ERROR 1
6 #define OK 0
7
8
9 int main() {
10     pid_t first_ch, second_ch;
11     int status, data;
12
13     first_ch = fork();
14
15     if (first_ch == - 1) {
16         perror("Can't fork.\n ");
17         return FORK_ERROR;
18     } else if (first_ch == 0) {
19         sleep(2);
20         fprintf(stdout, "\n1st child process:\tpid = %d\tppid = %d\tgroup
id = %d\n",
21             getpid(), getppid(), getpgrp());
22         puts("\nCur date:");
23
24         if (execlp("date", "", NULL) == - 1) {
25             perror("error exec");
26         }
```



```

27
28
29     return OK;
30 } else {
31     data = wait(&status);
32
33     if (WIFEXITED(status)) {
34         fprintf(stdout, "\nChild process (%d) finished with code %d\n",
35             data, WEXITSTATUS(status));
36     } else if (WIFSIGNALED(status)) {
37         fprintf(stdout, "\nChild process (%d) finished from signal
38 with code %d\n",
39             data, WTERMSIG(status));
40     } else if (WIFSTOPPED(status)) {
41         fprintf(stdout, "\nChild process (%d) finished from signal
42 with code %d\n",
43             data, WSTOPSIG(status));
44     }
45
46     second_ch = fork();
47
48     if (second_ch == - 1) {
49         perror("\nCan't fork.\n ");
50         return FORK_ERROR;
51     } else if (second_ch == 0) {
52         sleep(2);
53         fprintf(stdout, "\n2nd child process:\tpid = %d\tppid = %d\t
54 tgroup id = %d\n",
55             getpid(), getppid(), getpgrp());
56         puts("\nps al:");
57
58         if (execlp("ps" , "-al" , NULL) == - 1) {
59             perror("error exec");
60         }
61
62         return OK;
63     }
64
65     data = wait(&status);
66
67     if (WIFEXITED(status)) {
68         fprintf(stdout, "\nChild process (%d) finished with code %d\n",
69             data, WEXITSTATUS(status));
70     } else if (WIFSIGNALED(status)) {
71         fprintf(stdout, "\nChild process (%d) finished from signal
72 with code %d\n",

```

```

69         data, WTERMSIG(status));
70     } else if (WIFSTOPPED(status)) {
71         fprintf(stdout, "\nChild process %d finished from signal with
code %d\n",
72             data, WSTOPSIG(status));
73     }
74
75     fprintf(stdout, "\nParent process:\tpid = %d\tchild proc. id = %d,
%d\tgroup id = %d\n",
76         getpid(), first_ch , second_ch, getpgrp());
77
78     return OK;
79 }
80
81 return OK;
82 }

```

### 3.3 Демонстрация работы программы

```

1st child process:      pid = 9011      ppid = 9010      group id = 9010

Cur date:
понедельник, 28 декабря 2020 г. 16:48:59 (+04)

Child process (9011) finished with code 0

2nd child process:      pid = 9012      ppid = 9010      group id = 9010

ps al:
  PID TTY          TIME CMD
  523 ttys000      0:00.36 /bin/zsh -l
  8981 ttys001      0:00.42 /bin/zsh
  9010 ttys001      0:00.00 ./a.out

Child process (9012) finished with code 0

Parent process: pid = 9010      child proc. id = 9011, 9012      group id = 9010

```

Рисунок 3.1

## 4 Задание №4

### 4.1 Задание

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

### 4.2 Код программы

Листинг 4.1: task4

```
0 #include <stdio.h>
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4
5 #define LEN_STR 50
6
7 #define FORK_ERROR 1
8 #define PIPE_ERROR 2
9 #define OK 0
10
11
12 int main() {
13     pid_t first_ch, second_ch;
14     int fd_array[2];
15     int status, data;
16     char str_w[LEN_STR];
17
18     if (pipe(fd_array) == -1) {
19         perror("Couldn't pipe");
20         return PIPE_ERROR;
21     }
22
23     first_ch = fork();
24
25     if (first_ch == - 1) {
26         perror("Can't fork.\n ");
27         return FORK_ERROR;
28     } else if (first_ch == 0) {
29         close(fd_array[0]);
30         if (!write(fd_array[1], "Child msg 1\n", 12)) {
31             perror("Can't write\n");
```

```

32         return PIPE_ERROR;
33     }
34
35     return OK;
36 } else {
37     data = wait(&status);
38
39     if (WIFEXITED(status)) {
40         fprintf(stdout, "Child process (%d) finished, code = %d\n",
41                 data, WEXITSTATUS(status));
42     } else if (WIFSIGNALED(status)) {
43         fprintf(stdout, "Child process (%d) finished from signal, code
44 = %d\n",
45                 data, WTERMSIG(status));
46     } else if (WIFSTOPPED(status)) {
47         fprintf(stdout, "Child process (%d) finished from signal, code
48 = %d\n",
49                 data, WSTOPSIG(status));
50     }
51
52     second_ch = fork();
53
54     if (second_ch == - 1) {
55         perror("Can't fork.\n ");
56         return FORK_ERROR;
57     } else if (second_ch == 0) {
58         close(fd_array[0]);
59         if (!write(fd_array[1], "Child msg 2\n", 12)) {
60             perror("Can't write\n");
61             return PIPE_ERROR;
62         }
63
64         return OK;
65     }
66
67     data = wait(&status);
68
69     if (WIFEXITED(status)) {
70         fprintf(stdout, "\nChild process (%d) finished, code = %d\n",
71                 data, WEXITSTATUS(status));
72     } else if (WIFSIGNALED(status)) {
73         fprintf(stdout, "Child process (%d) finished from signal, code
74 = %d\n",
75                 data, WTERMSIG(status));
76     } else if (WIFSTOPPED(status)) {
77         fprintf(stdout, "Child process (%d) finished from signal, code
78 = %d\n",
79                 data, WSTOPSIG(status));

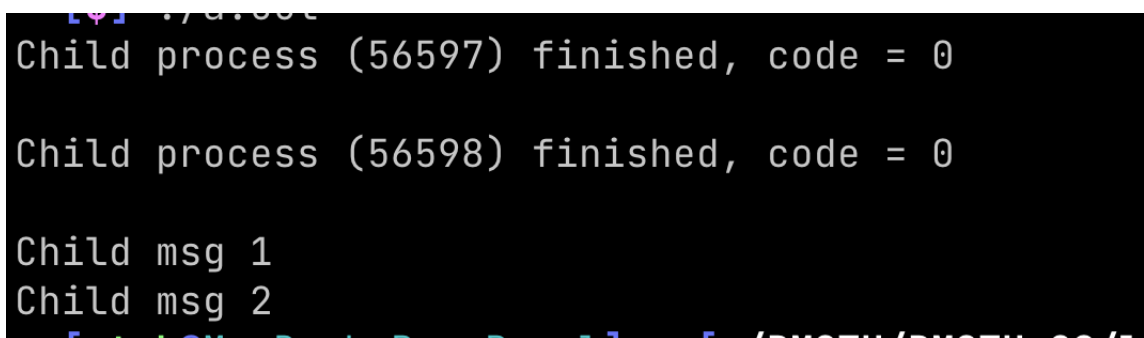
```

```

76     }
77
78     close(fd_array[1]);
79     if (read(fd_array[0], str_w, LEN_STR) < 0) {
80         perror("Can't read\n");
81         return PIPE_ERROR;
82     }
83
84     fprintf(stdout, "\n%s", str_w);
85
86     return OK;
87 }
88
89 return OK;
90 }

```

### 4.3 Демонстрация работы программы



```

[?] ./a.out
Child process (56597) finished, code = 0

Child process (56598) finished, code = 0

Child msg 1
Child msg 2
[?] ./a.out

```

Рисунок 4.1

## 5 Задание №5

### 5.1 Задание

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

В своей программе я написал обработчик сигналов SIGINT: сигнал прерывания Ctrl - C с терминала. А также SIGTSTP: сигнал прерывания Ctrl - Z с терминала.

### 5.2 Код программы

Листинг 5.1: task5

```
0 #include <stdio.h>
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4 #include <signal.h>
5
6 #define LEN_STR 50
7 #define TIME_SLEEP 30
8
9 #define FORK_ERROR 1
10 #define PIPE_ERROR 2
11 #define OK 0
12
13 int flag = 0;
14
15 void processing_sigint(int sig_data) {
16     fprintf(stdout, "\nProcess (%d), signal = %d\n", getpid(), sig_data);
17     flag = SIGINT;
18 }
19
20
21 void processing_sigtstp(int sig_data) {
22     fprintf(stdout, "\nProcess (%d), signal = %d\n", getpid(), sig_data);
23     flag = SIGTSTP;
24 }
25
```

```

26
27 int main() {
28     pid_t first_ch, second_ch;
29     int fd_array[2];
30     int status, data;
31     char str_w[LEN_STR];
32
33     signal(SIGTSTP, processing_sigtstp);
34
35     if (pipe(fd_array) == -1) {
36         perror("Couldn't pipe");
37         return PIPE_ERROR;
38     }
39
40     puts("Press the Ctrl + Z to continue");
41     sleep(TIME_SLEEP);
42
43     first_ch = fork();
44
45     if (first_ch == - 1) {
46         perror("Can't fork.\n ");
47         return FORK_ERROR;
48     } else if (first_ch == 0) {
49         if (flag == SIGTSTP) {
50             close(fd_array[0]);
51             if (!write(fd_array[1], "Child msg 1\n", 12)) {
52                 perror("Can't write\n");
53                 return PIPE_ERROR;
54             }
55         }
56
57         return OK;
58     } else {
59         data = wait(&status);
60
61         if (WIFEXITED(status)) {
62             fprintf(stdout, "Child process (%d) finished, code = %d\n",
63                     data, WEXITSTATUS(status));
64         } else if (WIFSIGNALED(status)) {
65             fprintf(stdout, "Child process (%d) finished from signal, code
66 = %d\n",
67                     data, WTERMSIG(status));
68         } else if (WIFSTOPPED(status)) {
69             fprintf(stdout, "Child process (%d) finished from signal, code
70 = %d\n",
71                     data, WSTOPSIG(status));

```

```

72     second_ch = fork();
73
74     if (second_ch == - 1) {
75         perror("Can't fork.\n ");
76         return FORK_ERROR;
77     } else if (second_ch == 0) {
78         if (flag == SIGTSTP) {
79             close(fd_array[0]);
80             if (!write(fd_array[1], "Child msg 2\n", 12)) {
81                 perror("Can't write\n");
82                 return PIPE_ERROR;
83             }
84         }
85
86         return OK;
87     }
88
89     data = wait(&status);
90
91     if (WIFEXITED(status)) {
92         fprintf(stdout, "\nChild process (%d) finished, code = %d\n",
93             data, WEXITSTATUS(status));
94     } else if (WIFSIGNALED(status)) {
95         fprintf(stdout, "Child process (%d) finished from signal, code
96 = %d\n",
97             data, WTERMSIG(status));
98     } else if (WIFSTOPPED(status)) {
99         fprintf(stdout, "Child process (%d) finished from signal, code
100 = %d\n",
101             data, WSTOPSIG(status));
102     }
103
104     signal(SIGINT, processing_sigint);
105
106     puts("Press the Ctrl + C to get process information");
107     sleep(TIME_SLEEP);
108
109     if (flag == SIGINT) {
110         close(fd_array[1]);
111         if (read(fd_array[0], str_w, LEN_STR) < 0) {
112             perror("Can't read\n");
113             return PIPE_ERROR;
114         }
115
116         fprintf(stdout, "\n%s", str_w);
117     }
118
119     return OK;

```



```
118     }  
119  
120     return OK;  
121 }
```

### 5.3 Демонстрация работы программы

```
Press the Ctrl + Z to continue  
^Z  
Process (57191), signal = 18  
Child process (57202) finished, code = 0  
  
Child process (57203) finished, code = 0  
Press the Ctrl + C to get process information  
^C  
Process (57191), signal = 2  
  
Child msg 1  
Child msg 2
```

Рисунок 5.1