



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 дисциплине "Операционные системы"

Тема Процессы. Системные вызовы fork() и exec()

Студент Шацкий Р.Е.

Группа ИУ7-55Б

Оценка (баллы) \_\_\_\_\_

Преподаватели Рязанова Н.Ю.

## Задание 1. Процессы-сироты.

В программе создаются не менее двух потомков. В потомках вызывается `sleep()`. Чтобы предок гарантированно завершился раньше своих потомков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе. Продемонстрировать «усыновление». Для этого надо в потомках вывести идентификаторы: собственный, предка, группы до блокировки и после блокировки.

Листинг 1: Процессы-сироты

```
1      #include <stdio.h>
2      #include <stdlib.h>
3
4      enum error_t {
5          no_error,
6          fork_failure
7      };
8
9      #define PROC_COUNT 3
10     #define SLEEP_TIME 1
11
12     int main() {
13         int children[PROC_COUNT];
14         printf("\n Предок — PID: %d, GROUP: %d\n", getpid(), getpgrp());
15
16         for (int i = 0; i < PROC_COUNT; ++i) {
17             int child_pid = fork();
18
19             if (child_pid == -1) {
20                 perror("Ошибка fork\n");
21                 return fork_failure;
22             }
23             else if (child_pid == 0) {
24                 printf("Для потомка до усыновления      %d — PID: %d, PPID: %d,\n",
25                     GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
26                 sleep(SLEEP_TIME * 4);
27                 printf("Для потомка после усыновления    %d — PID: %d, PPID: %\n",
28                     d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
29                 return no_error;
30             }
31             else {
32                 sleep(SLEEP_TIME);
33                 children[i] = child_pid;
34             }
35         }
36
37         printf("Процессы, созданные предком:\n");
38         for (int i = 0; i < PROC_COUNT; ++i) {
39             printf("Потомок      %d — PID: %d, ", i + 1, children[i]);
```

```
39     printf("\b\b\nПредок завершился\n");
40
41     return no_error;
42 }
```

img/task\_01.png

Рис. 1: Демонстрация работы программы (задание №1).

## Задание 2.

Предок ждет завершения своих потомков, используя системный вызов `wait()`. Вывод соответствующих сообщений на экран. В программе необходимо, чтобы предок выполнял анализ кодов завершения потомков.

Листинг 2: `wait()`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <wait.h>
4
5  enum error_t {
6      no_error,
7      fork_failure
8  };
9
10 #define PROC_COUNT 3
11 #define SLEEP_TIME 1
12
13 int main() {
14     int children[PROC_COUNT];
15     printf("\nПредок — PID: %d, GROUP: %d\n", getpid(), getppid());
16
17     for (int i = 0; i < PROC_COUNT; ++i) {
18         int child_pid = fork();
19
20         if (child_pid == -1) {
21             perror("Can't fork");
22             return fork_failure;
23         }
24         else if (child_pid == 0) {
25             sleep(SLEEP_TIME);
26             printf("Для потомка    %d — PID: %d, PPID: %d, GROUP: %d\n", i +
27                 1, getpid(), getppid(), getppid());
28             return 0;
29         }
30         else {
31             children[i] = child_pid;
32         }
33     }
34
35     printf("Процессы, созданные предком:\n");
36     for (int i = 0; i < PROC_COUNT; ++i) {
37         int status;
38         int stat_value = 0;
39
40         pid_t child_pid = wait(&status);
41         printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
42             status);
```

```

41     if (WIFEXITED(stat_value)) {
42         printf("\tПотомок завершился нормально. Код завершения: %d\n",
43             WEXITSTATUS(stat_value));
44     }
45     else if (WIFSIGNALED(stat_value)) {
46         printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
47             нала: %d\n", WTERMSIG(stat_value));
48     }
49     else if (WIFSTOPPED(stat_value)) {
50         printf("\tПотомок остановился. Номер сигнала: %d\n", WSTOPSIG(
51             stat_value));
52     }
53     printf("Предок завершился\n");
54     return no_error;
55 }

```

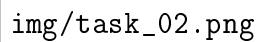
The image is a placeholder for a screenshot, indicated by the text 'img/task\_02.png'. It is intended to show the output of a program for task 2.

Рис. 2: Демонстрация работы программы (задание №2).

### Задание 3.

Потомки переходят на выполнение других программ, которые передаются системному вызову `exec()` в качестве параметра. Потомки должны выполнять разные программы. Преродок ждет завершения своих потомков с анализом кодов завершения. На экран выводятся соответствующие сообщения.

Листинг 3: `exec()`

```
1 | #include <stdio.h>
```

```

2  #include <stdlib.h>
3  #include <wait.h>
4  #include <string.h>
5
6  enum error_t {
7      no_error,
8      fork_fail,
9      exec_fail
10 };
11
12 #define PROC_COUNT 3
13 #define SLEEP_TIME 2
14
15 int main() {
16     int children[PROC_COUNT];
17     char *commands[PROC_COUNT] = {"ls", "ps", "pwd"};
18     char *args[PROC_COUNT] = {"-a", "ax", "-L"};
19     printf("Предок — PID: %d, GROUP: %d\n", getpid(), getpgrp());
20
21     for (int i = 0; i < PROC_COUNT; ++i) {
22         int child_pid = fork();
23
24         if (child_pid == -1) {
25             perror("Ошибка fork'a");
26             return fork_fail;
27         }
28         else if (child_pid == 0) {
29             sleep(SLEEP_TIME);
30             printf("Для потомка %d — PID: %d, PPID: %d, GROUP: %d\n", i +
31                 1, getpid(), getppid(), getpgrp());
32
33             int res = execlp(commands[i], args[i], 0);
34             if (res == -1) {
35                 perror("Еxec невозможен");
36                 return exec_fail;
37             }
38
39             return no_error;
40         }
41         else {
42             children[i] = child_pid;
43         }
44     }
45
46     printf("Процессы, созданные предком:\n");
47     for (int i = 0; i < PROC_COUNT; ++i) {
48         int status;
49         int stat_value = 0;
50
51         pid_t child_pid = wait(&status);


```

```

51     printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
52           status);
53
54     if (WIFEXITED(stat_value)) {
55         printf("\tПотомок завершился нормально. Код завершения: %d\n",
56               WEXITSTATUS(stat_value));
57     }
58     else if (WIFSIGNALED(stat_value)) {
59         printf("\tПотомок завершился перехвачиваемым сигналом. Номер сиг-
60               нала: %d\n", WTERMSIG(stat_value));
61     }
62     else if (WIFSTOPPED(stat_value)) {
63         printf("\tПотомок остановился. Номер сигнала: %d\n", WSTOPSIG(
64               stat_value));
65     }
66     printf("Предок завершился\n");
67
68     return no_error;
69 }

```





img/task\_03.png

Рис. 3: Демонстрация работы программы (задание №3).

## Задание 4.

Предок и потомки обмениваются сообщениями через неименованный программный канал. Причем оба потомка пишут свои сообщения в один программный канал, а предок их считывает из канала. Потомки должны посылать предку разные сообщения по содержанию и размеру. Предок считывает сообщения от потомков и выводит их на экран. Предок ждет завершения своих потомков и анализирует код их завершения. Вывод соответствующих сообщений на экран.

#### Листинг 4: pipe()

```

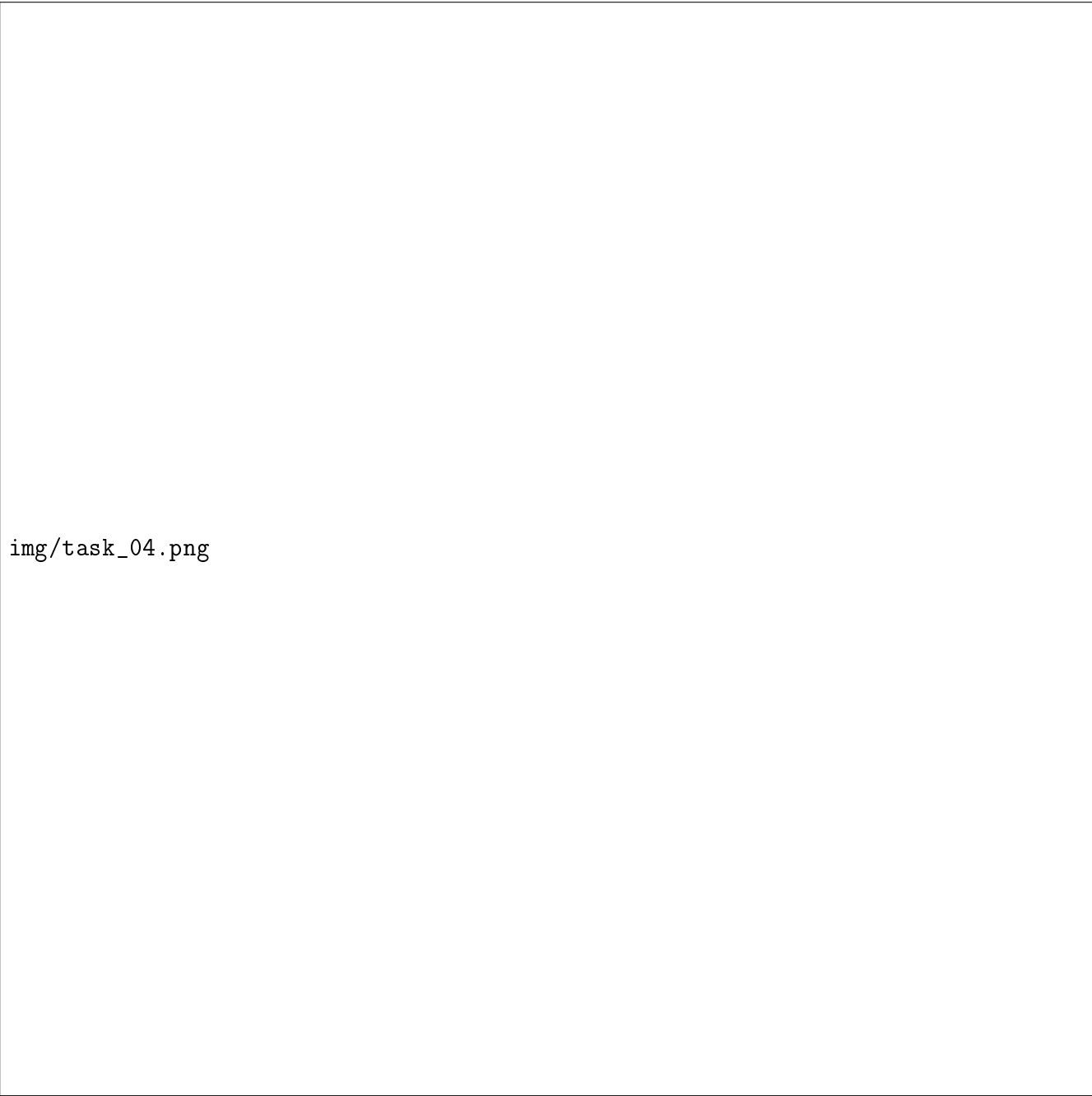
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <wait.h>
4  #include <string.h>
5
6  enum error_t {
7      no_error,
8      fork_fail,
9      exec_fail,
10     pipe_fail
11 };
12
13 #define PROC_COUNT 3
14 #define SLEEP_TIME 2
15 #define STR_BUFF_SIZE 64
16
17 int main() {
18     int pipefd[2]; // [0] — чтение, [1] — запись
19     if (pipe(pipefd) == -1) {
20         perror("Ошибка pipe");
21         return pipe_fail;
22     }
23
24     char *msgs[PROC_COUNT] = {"First msg\n", "Second msg\n", "Third
        message\n"};
25     char str_buff[STR_BUFF_SIZE] = {0};
26
27     int children[PROC_COUNT];
28     printf("Предок — PID: %d, GROUP: %d\n", getpid(), getpgrp());
29
30     for (int i = 0; i < PROC_COUNT; ++i) {
31         int child_pid = fork();
32
33         if (child_pid == -1) {
34             perror("Ошибка fork'a");
35             return fork_fail;
36         }
37         else if (child_pid == 0) {
38             printf("Для потомка %d — PID: %d, PPID: %d, GROUP: %d\n", i +
                1, getpid(), getppid(), getpgrp());
39
40             close(pipefd[0]);
41             write(pipefd[1], msgs[i], strlen(msgs[i]));
42
43             printf("Сообщение %d отправлено потомку\n\n", i + 1);
44
45             return no_error;
46         }

```

```

47     else {
48         children[i] = child_pid;
49     }
50 }
51
52 printf("Процессы, созданные предком:\n");
53 for (int i = 0; i < PROC_COUNT; ++i) {
54     int status;
55     int stat_value = 0;
56
57     pid_t child_pid = wait(&status);
58     printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
59           status);
60
61     if (WIFEXITED(stat_value)) {
62         printf("\tПотомок завершился нормально. Код завершения: %d\n\n",
63               WEXITSTATUS(stat_value));
64     }
65     else if (WIFSIGNALED(stat_value)) {
66         printf("\tПотомок завершился перехвачиваемым сигналом. Номер сиг
67               нала: %d\n\n", WTERMSIG(stat_value));
68     }
69     else if (WIFSTOPPED(stat_value)) {
70         printf("\tПотомок остановился. Номер сигнала: %d\n\n", WSTOPSIG(
71               stat_value));
72     }
73 }
74
75 close(pipefd[1]);
76 read(pipefd[0], str_buff, STR_BUFF_SIZE);
77 printf("Сообщения, полученные предком: %s", str_buff);
78
79 printf("Предок завершился\n");
80 return no_error;
81 }

```



img/task\_04.png

Рис. 4: Демонстрация работы программы (задание №4).

## Задание 5.

Предок и потомки аналогично №5 обмениваются сообщениями через неименованный программный канал. В программу включается собственный обработчик сигнала. С помощью сигнала меняется ход выполнения программы. При получении сигнала потомки записывают сообщения в канал, если сигнал не поступает, то не записывают. Предок ждет завершения своих потомков и анализирует коды их завершений. Вывод соответствующих сообщений на экран.

Листинг 5: signal()

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <wait.h>
4  #include <string.h>
5  #include <signal.h>
6
7  enum error_t {
8      no_error,
9      fork_fail,
10     exec_fail,
11     pipe_fail
12 };
13
14 #define PROC_COUNT 3
15 #define SLEEP_TIME 2
16 #define STR_BUFF_SIZE 64
17
18 void do_nothing(int sigint) {
19     printf("Ничего не происходит?\n");
20 }
21
22 static int sig_status = 0;
23
24 void inc_status(int sigint) {
25     sig_status++;
26     printf("Статус увеличен — %d\n", sig_status);
27 }
28
29 int main() {
30     int pipefd[2]; // [0] — чтение, [1] — запись
31     if (pipe(pipefd) == -1) {
32         perror("Ошибка pipe\n");
33         return pipe_fail;
34     }
35
36     char *msgs[PROC_COUNT] = {"First msg\n", "Second msg\n", "Third
37     message\n"};
38     char str_buff[STR_BUFF_SIZE] = {0};
39
40     int children[PROC_COUNT];
41     printf("Предок — PID: %d, GROUP: %d\n", getpid(), getpgrp());
42     signal(SIGINT, do_nothing);
43
44     for (int i = 0; i < PROC_COUNT; ++i) {
45         int child_pid = fork();
46
47         if (child_pid == -1) {
48             perror("Ошибка fork'a");

```

```

48     return fork_fail;
49 }
50 else if (child_pid == 0) {
51     printf("Для потомка      %d — PID: %d, PPID: %d, GROUP: %d\n", i +
52           1, getpid(), getppid(), getpgrp());
53
54     signal(SIGINT, inc_status);
55     sleep(SLEEP_TIME);
56
57     if (sig_status != 0) {
58         close(pipefd[0]);
59         write(pipefd[1], msgs[i], strlen(msgs[i]));
60         printf("Сообщение      %d отправлено потомку\n\n", i + 1);
61     }
62     else {
63         printf("Сигнала не было\n\n");
64     }
65
66     return no_error;
67 }
68 else {
69     children[i] = child_pid;
70 }
71 }
72
73 printf("Процессы, созданные предком:\n");
74 for (int i = 0; i < PROC_COUNT; ++i) {
75     int status;
76     int stat_value = 0;
77
78     pid_t child_pid = wait(&status);
79     printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
80           status);
81
82     if (WIFEXITED(stat_value)) {
83         printf("\tПотомок завершился нормально. Код завершения: %d\n\n",
84               WEXITSTATUS(stat_value));
85     }
86     else if (WIFSIGNALED(stat_value)) {
87         printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
88               нала: %d\n\n", WTERMSIG(stat_value));
89     }
90     else if (WIFSTOPPED(stat_value)) {
91         printf("\tПотомок остановился. Номер сигнала: %d\n\n", WSTOPSIG(
92               stat_value));
93     }
94 }
95
96 close(pipefd[1]);
97 read(pipefd[0], str_buff, STR_BUFF_SIZE);

```

```
93     printf("Сообщения, полученные предком: %s", str_buff);  
94  
95     printf("Предок завершился\n");  
96     return no_error;  
97 }
```

img/task\_05.png

Рис. 5: Демонстрация работы программы (задание №5).