



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.

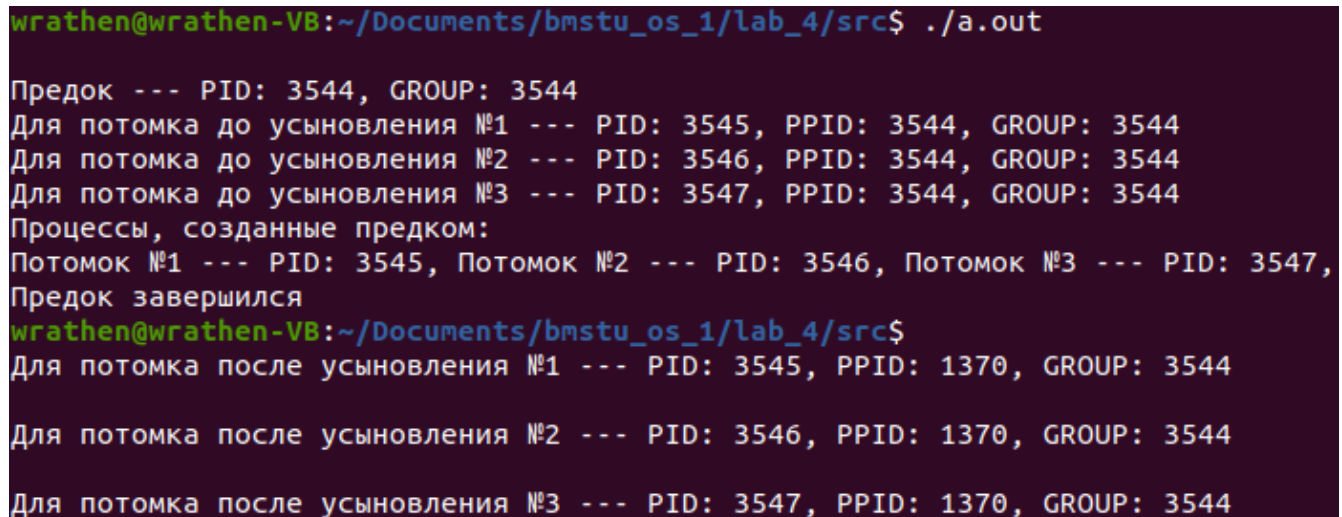
Листинг 1: Процессы-сироты

```
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 No%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 No%d — PID: %d, PPID: %d\n",
28                     , 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 No%d — PID: %d, ", i + 1, children[i]);
```

```

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

```



```

wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 3544, GROUP: 3544
Для потомка до усыновления №1 --- PID: 3545, PPID: 3544, GROUP: 3544
Для потомка до усыновления №2 --- PID: 3546, PPID: 3544, GROUP: 3544
Для потомка до усыновления №3 --- PID: 3547, PPID: 3544, GROUP: 3544
Процессы, созданные предком:
Потомок №1 --- PID: 3545, Потомок №2 --- PID: 3546, Потомок №3 --- PID: 3547,
Предок завершился
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$
Для потомка после усыновления №1 --- PID: 3545, PPID: 1370, GROUP: 3544
Для потомка после усыновления №2 --- PID: 3546, PPID: 1370, GROUP: 3544
Для потомка после усыновления №3 --- PID: 3547, PPID: 1370, GROUP: 3544

```

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

Задание 2.

Предок ждет завершения своих потомком, 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(), getpgrp());
16
17     for (int i = 0; i < PROC_COUNT; ++i) {
18         int child_pid = fork();

```

```

19     if (child_pid == -1) {
20         perror("Can't fork");
21         return fork_failure;
22     }
23     else if (child_pid == 0) {
24         sleep(SLEEP_TIME);
25         printf("Для потомка No%d — PID: %d, PPID: %d, GROUP: %d\n", i +
26             1, getpid(), getppid(), getpgrp());
27         return 0;
28     }
29     else {
30         children[i] = child_pid;
31     }
32 }
33
34 printf("Процессы, созданные предком:\n");
35 for (int i = 0; i < PROC_COUNT; ++i) {
36     int status;
37     int stat_value = 0;
38
39     pid_t child_pid = wait(&status);
40     printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
41         status);
42
43     if (WIFEXITED(stat_value)) {
44         printf("\tПотомок завершился нормально. Код завершения: %d\n",
45             WEXITSTATUS(stat_value));
46     }
47     else if (WIFSIGNALED(stat_value)) {
48         printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
49             нала: %d\n", WTERMSIG(stat_value));
50     }
51     else if (WIFSTOPPED(stat_value)) {
52         printf("\tПотомок остановился. Номер сигнала: %d\n", WSTOPSIG(
53             stat_value));
54     }
55 }
56 printf("Предок завершился\n");
57
58 return no_error;
59 }

```

```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out

Предок --- PID: 3636, GROUP: 3636
Процессы, созданные предком:
Для потомка №1 --- PID: 3637, PPID: 3636, GROUP: 3636
Потомок с PID = 3637 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0
Для потомка №2 --- PID: 3638, PPID: 3636, GROUP: 3636
Потомок с PID = 3638 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0
Для потомка №3 --- PID: 3639, PPID: 3636, GROUP: 3636
Потомок с PID = 3639 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0
Предок завершился
```

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. Предок ждет завершения своих потомков с анализом кодов завершения. На экран выводятся соответствующие сообщения.

С помощью вызова `exec()` выполняются программы из первых лабораторных работ на курсе по языку C:

Определить нормальный вес человека и индекс массы его тела по формулам: $h * t / 240$ и m / h^2 , где h - рост человека (измеряемый в сантиметрах в первой формуле и в метрах - во второй); t - длина окружности грудной клетки (в сантиметрах); m - вес (в килограммах). Порядок ввода параметров: h , t , m (h в см, t в см, m в кг).

Листинг 3: Код программы `weight_index`

```
1  #include <stdio.h>
2
3  int main(int argc, char* argv[])
4  {
5      for (int i = 0; i < argc; ++i) {
6          printf("Аргумент No%d = %s\n", i, argv[i]);
7      }
8
9      float h, t, m;
10
11     printf("Введите через пробел рост в см, длину окр. грудной клетки в см
        и вес в кг:\n");
12     scanf("%f %f %f", &h, &t, &m);
13
```

```

14     float weight = h * t / 240;
15     float index = m / (h / 100) / (h / 100);
16
17     printf("%.5f %.5f\n", weight, index);
18
19     return 0;
20 }

```

Треугольник задан координатами вершин. Определить тип треугольника. Ввод: x1, y1, x2, y2, x3, y3. Вывод: 0 - остроугольный, 1 - прямоугольный, 2 - тупоугольный.

Листинг 4: Код программы triangle_type

```

1  #include <stdio.h>
2  #include <math.h>
3
4  int check(float a2, float b2, float c2)
5  {
6      float h = 1e-6;
7      int result = 0;
8      /* 90 degrees check */
9      if (fabsf(a2 - b2 - c2) < h || fabsf(b2 - a2 - c2) < h || fabsf(c2 -
10         b2 - a2) < h)
11      {
12          result = 1;
13      }
14      /* >90 degrees check */
15      else if (b2 + c2 - a2 < 0 || b2 + a2 - c2 < 0 || a2 + c2 - b2 < 0)
16      {
17          result = 2;
18      }
19      return result;
20 }
21
22 int main(int argc, char* argv[])
23 {
24     for (int i = 0; i < argc; ++i) {
25         printf("Аргумент No%d = %s\n", i, argv[i]);
26     }
27
28     printf("Введите координаты вершин треугольника:\n");
29     float x1, y1, x2, y2, x3, y3;
30     if (scanf("%f %f %f %f %f %f", &x1, &y1, &x2, &y2, &x3, &y3) != 6)
31     {
32         printf("Input error");
33         return -1;
34     }
35     /* is it triangle check */
36     if (fabsf((x1 - x3) * (y2 - y3) - (x2 - x3) * (y1 - y3)) < 1e-6)
37     {
38         printf("Input error");

```

```

38     return -1;
39 }
40
41 float a2 = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
42 float b2 = (x3 - x2) * (x3 - x2) + (y3 - y2) * (y3 - y2);
43 float c2 = (x3 - x1) * (x3 - x1) + (y3 - y1) * (y3 - y1);
44
45 printf("%d\n", check(a2, b2, c2));
46 return 0;
47 }

```

Листинг 5: 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] = { "./weight_index.out", "./triangle_type.
18         out" };
19     char *args[PROC_COUNT] = { "ПЕРВЫЙ", "__ВТОРОЙ__", "" };
20     printf("Предок — PID: %d, GROUP: %d\n", getpid(), getpgrp());
21
22     for (int i = 0; i < PROC_COUNT; ++i) {
23         int child_pid = fork();
24
25         if (child_pid == -1) {
26             perror("Ошибка fork'a");
27             return fork_fail;
28         }
29         else if (child_pid == 0) {
30             sleep(SLEEP_TIME);
31             printf("Для потомка No%d — PID: %d, PPID: %d, GROUP: %d\n", i +
32                 1, getpid(), getppid(), getpgrp());
33
34             int res = execlp(commands[i], args[i], 0);
35             if (res == -1) {
36                 perror("Еxec невозможен");
37                 return exec_fail;
38             }
39         }
40     }
41 }

```

```

36         }
37
38         return no_error;
39     }
40     else {
41         children[i] = child_pid;
42     }
43 }
44
45 printf("Процессы, созданные предком:\n");
46 for (int i = 0; i < PROC_COUNT; ++i) {
47     int status;
48     int stat_value = 0;
49
50     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 неперехватываемым 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 }
67 printf("Предок завершился\n");
68
69 return no_error;
70 }

```



```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 4491, GROUP: 4491
Процессы, созданные предком:
Для потомка №1 --- PID: 4492, PPID: 4491, GROUP: 4491
Аргумент №0 = ПЕРВЫЙ
Введите через пробел рост в см, длину окр. грудной клетки в см и вес в кг:
Для потомка №2 --- PID: 4493, PPID: 4491, GROUP: 4491
Аргумент №0 = __ВТОРОЙ__
Введите координаты вершин треугольника:
180 50 78
37.50000 24.07408
Потомок с PID = 4492 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0
0 0 0 5 6 0
1
Потомок с PID = 4493 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0
Предок завершился
```

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 6: 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 No%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 No%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],
    status);
59
60      if (WIFEXITED(stat_value)) {
61          printf("\tПотомок завершился нормально. Код завершения: %d\n\n",
    WEXITSTATUS(stat_value));
62      }
63      else if (WIFSIGNALED(stat_value)) {
64          printf("\tПотомок завершился перекрываваемым сигналом. Номер сиг

```

```

        нала: %d\n\n", WTERMSIG(stat_value));
65     }
66     else if (WIFSTOPPED(stat_value)) {
67         printf("\tПотомок остановился. Номер сигнала: %d\n\n", WSTOPSIG(
            stat_value));
68     }
69 }
70
71 close(pipefd[1]);
72 read(pipefd[0], str_buff, STR_BUFF_SIZE);
73 printf("Сообщения, полученные предком: %s", str_buff);
74
75 printf("Предок завершился\n");
76 return no_error;
77 }

```

```

wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 3741, GROUP: 3741
Процессы, созданные предком:
Для потомка №2 --- PID: 3743, PPID: 3741, GROUP: 3741
Сообщение №2 отправлено потомку

Потомок с PID = 3742 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0

Для потомка №3 --- PID: 3744, PPID: 3741, GROUP: 3741
Сообщение №3 отправлено потомку

Потомок с PID = 3743 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0

Для потомка №1 --- PID: 3742, PPID: 3741, GROUP: 3741
Сообщение №1 отправлено потомку

Потомок с PID = 3744 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0

Сообщения, полученные предком: Second msg
Third message
First msg
Предок завершился

```

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 7: 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 int children[PROC_COUNT];
40 printf("Предок — PID: %d, GROUP: %d\n", getpid(), getpgrp());
41 signal(SIGINT, do_nothing);
42
43 for (int i = 0; i < PROC_COUNT; ++i) {
44     int child_pid = fork();
45
46     if (child_pid == -1) {
47         perror("Ошибка fork'a");
48         return fork_fail;
49     }
50     else if (child_pid == 0) {
51         printf("Для потомка No%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 No%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));

```

```

85     }
86     else if (WIFSTOPPED(stat_value)) {
87         printf("\tПотомок остановился. Номер сигнала: %d\n\n", WSTOPSIG(
            stat_value));
88     }
89 }
90
91 close(pipefd[1]);
92 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 }

```

```

wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 3791, GROUP: 3791
Процессы, созданные предком:
Для потомка №2 --- PID: 3793, PPID: 3791, GROUP: 3791
Для потомка №3 --- PID: 3794, PPID: 3791, GROUP: 3791
Для потомка №1 --- PID: 3792, PPID: 3791, GROUP: 3791
Сигнала не было

Потомок с PID = 3792 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0

Сигнала не было

Потомок с PID = 3793 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0

Сигнала не было

Потомок с PID = 3794 завершился. Статус: 0
    Потомок завершился нормально. Код завершения: 0

Сообщения, полученные предком: Предок завершился

```

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