



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 Зайцева А. А.

Группа ИУ7-52Б

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: Код программы к заданию №1

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5 #define RET_OK 0
6 #define RET_ERR_FORK 1
7
8 #define FORK_OK 0
9 #define FORK_ERR -1
10
11 #define INTERVAL 1
12
13 int main()
14 {
15     pid_t childpid1, childpid2;
16     if ((childpid1 = fork()) == FORK_ERR)
17     {
18         perror("Can't fork first child process.\n");
19         return RET_ERR_FORK;
20     }
21     else if (childpid1 == FORK_OK)
22     {
23         printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
24             getpid(), getppid(), getpgrp());
25
26         sleep(INTERVAL);
27         printf("First child process (has become an orphan): pid = %d, ppid = %d,
28             pgrp = %d\n",
29             getpid(), getppid(), getpgrp());
30
31         printf("First child process is dead now\n");
32
33         exit(RET_OK);
34     }
35
36     if ((childpid2 = fork()) == FORK_ERR)
37     {
38         perror("Can't fork second child process.\n");
39         return RET_ERR_FORK;
```

```

40 }
41 else if (childpid2 == FORK_OK)
42 {
43     printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
44         getpid(), getppid(), getpgrp());
45
46     sleep(INTERVAL);
47     printf("Second child process (has become an orphan): pid = %d, ppid = %d
48         , pgrp = %d\n",
49         getpid(), getppid(), getpgrp());
50
51     printf("Second child process is dead now\n");
52     exit(RET_OK);
53 }
54
55 printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n",
56     getpid(), getpgrp(), childpid1, childpid2);
57 printf("Parent process is dead now\n");
58 return RET_OK;
59 }

```

```

alena@DESKTOP-TJ9D65N:~/lab4$ ./task1
First child process: pid = 285, ppid = 284, pgrp = 284
Parent process: pid = 284, pgrp = 284, childpid1 = 285, childpid2 = 286
Second child process: pid = 286, ppid = 284, pgrp = 284
Parent process is dead now
alena@DESKTOP-TJ9D65N:~/lab4$ First child process (has become an orphan): pid = 285, ppid = 1, pgrp = 284
First child process is dead now
Second child process (has become an orphan): pid = 286, ppid = 1, pgrp = 284
Second child process is dead now

```

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: Код программы к заданию №2

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5
6 #define RET_OK 0
7 #define RET_ERR_FORK 1
8

```

```

9  #define FORK_OK 0
10 #define FORK_ERR -1
11
12 #define INTERVAL 1
13
14 int main()
15 {
16     pid_t childpid1, childpid2, childpid;
17     if ((childpid1 = fork()) == FORK_ERR)
18     {
19         perror("Can't fork first child process.\n");
20         return RET_ERR_FORK;
21     }
22     else if (childpid1 == FORK_OK)
23     {
24         printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
25             getpid(), getppid(), getpgrp());
26
27         exit(RET_OK);
28     }
29
30     if ((childpid2 = fork()) == FORK_ERR)
31     {
32         perror("Can't fork second child process.\n");
33         return RET_ERR_FORK;
34     }
35     else if (childpid2 == FORK_OK)
36     {
37         printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
38             getpid(), getppid(), getpgrp());
39
40         exit(RET_OK);
41     }
42
43     sleep(INTERVAL);
44     printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n",
45         getpid(), getpgrp(), childpid1, childpid2);
46
47     int ch_status;
48     for (int i = 0; i < 2; i++)
49     {
50         childpid = wait(&ch_status);
51         printf("Child with pid = %d has finished with status %d\n", childpid,
52             ch_status);
53
54         if (WIFEXITED(ch_status))
55             printf("Child exited normally with exit code %d\n", WEXITSTATUS(
56                 ch_status));
57         else if (WIFSIGNALED(ch_status))

```

```

56     printf("Child process ended with a non-intercepted signal number %d\n",
           WTERMSIG(ch_status));
57     else if (WIFSTOPPED(ch_status))
58     printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
           );
59 }
60
61 printf("Parent process is dead now\n");
62 return RET_OK;
63 }

```

```

alena@DESKTOP-TJ9D65N:~/lab4$ ./task2
First child process: pid = 70, ppid = 69, pgrp = 69
Second child process: pid = 71, ppid = 69, pgrp = 69
Parent process: pid = 69, pgrp = 69, childpid1 = 70, childpid2 = 71
Child with pid = 70 has finished with status 0
Child exited normally with exit code 0
Child with pid = 71 has finished with status 0
Child exited normally with exit code 0
Parent process is dead now

```

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

Листинг 3: Код программы к заданию №3

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5
6 #define RET_OK 0
7 #define RET_ERR_FORK 1
8 #define RET_CANT_EXECLP 2
9
10 #define FORK_OK 0
11 #define FORK_ERR -1
12
13 #define INTERVAL 1
14
15 int main()

```

```

16 {
17     pid_t childpid1, childpid2, childpid;
18     if ((childpid1 = fork()) == FORK_ERR)
19     {
20         perror("Can't fork first child process.\n");
21         return RET_ERR_FORK;
22     }
23     else if (childpid1 == FORK_OK)
24     {
25         printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
26             getpid(), getppid(), getpgrp());
27         if (execl("./task3_sum", "task3_sum", "2", "3", NULL) < 0)
28         {
29             perror("Can't execl from first child.\n");
30             exit(RET_CANT_EXECLP);
31         }
32         exit(RET_OK);
33     }
34
35     if ((childpid2 = fork()) == FORK_ERR)
36     {
37         perror("Can't fork second child process.\n");
38         return RET_ERR_FORK;
39     }
40     else if (childpid2 == FORK_OK)
41     {
42         sleep(INTERVAL);
43         printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
44             getpid(), getppid(), getpgrp());
45         if (execlp("./task3_write", "task3_write", "file_to_write.txt", "This
46             is test info", NULL) < 0)
47         {
48             perror("Can't execlp from second child.\n");
49             exit(RET_CANT_EXECLP);
50         }
51         exit(RET_OK);
52     }
53
54     sleep(INTERVAL * 2);
55     printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
56         d\n",
57         getpid(), getpgrp(), childpid1, childpid2);
58
59     int ch_status;
60     for (int i = 0; i < 2; i++)
61     {
62         childpid = wait(&ch_status);
63         printf("Child with pid = %d has finished with status %d\n", childpid,
64             ch_status);

```

```

63
64     if (WIFEXITED(ch_status))
65         printf("Child exited normally with exit code %d\n", WEXITSTATUS(
            ch_status));
66     else if (WIFSIGNALED(ch_status))
67         printf("Child process ended with a non-intercepted signal number %d\n",
            WTERMSIG(ch_status));
68     else if (WIFSTOPPED(ch_status))
69         printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
            );
70 }
71
72 printf("Parent process is dead now\n");
73 return RET_OK;
74 }

```

Листинг 4: Код вызываемой из первого потомка программы

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define RET_OK 0
5  #define RET_ERR_PARAM -1
6
7  int main(int argc, char **argv)
8  {
9      int a, b;
10
11     printf("This is program called from first child\n");
12
13     if ((argc != 3) ||
14         ((a = atoi(argv[1])) <= 0) ||
15         ((b = atoi(argv[2])) <= 0))
16     {
17         printf("Error: got wrong arguments\n");
18         return RET_ERR_PARAM;
19     }
20
21     printf("I can count sum: %d + %d = %d\n", a, b, (a + b));
22     return RET_OK;
23 }

```

Листинг 5: Код вызываемой из второго потомка программы

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  #define RET_OK 0
6  #define RET_ERR_PARAM -1
7  #define RET_ERR_FILE -2

```

```

8
9 int main(int argc, char **argv)
10 {
11     printf("This is program called from second child\n");
12
13     if (argc != 3)
14     {
15         printf("Error: got wrong arguments\n");
16         return RET_ERR_PARAM;
17     }
18
19     FILE *f = fopen(argv[1], "w");
20     if (!f)
21     {
22         printf("Error: cant open file\n");
23         return RET_ERR_FILE;
24     }
25
26     printf("I can write given info and info about myself here and to given
        file %s\n", argv[1]);
27
28     fprintf(f, "Given info: %s\n", argv[2]);
29     fprintf(stdout, "Given info: %s\n", argv[2]);
30
31     fprintf(f, "Info about myself: pid = %d, ppid = %d, pgrp = %d\n",
32     getpid(), getppid(), getpgrp());
33     fprintf(stdout, "Info about myself: pid = %d, ppid = %d, pgrp = %d\n",
34     getpid(), getppid(), getpgrp());
35
36     fclose(f);
37
38     return RET_OK;
39 }

```



```

alena@DESKTOP-TJ9D65N:~/lab4$ ./task3
First child process: pid = 390, ppid = 389, pgrp = 389
This is program called from first child
I can count sum: 2 + 3 = 5
Second child process: pid = 391, ppid = 389, pgrp = 389
This is program called from second child
I can write given info and info about myself here and to given file file_to_write.txt
Given info: This is test info
Info about myself: pid = 391, ppid = 389, pgrp = 389
Parent process: pid = 389, pgrp = 389, childpid1 = 390, childpid2 = 391
Child with pid = 390 has finished with status 0
Child exited normally with exit code 0
Child with pid = 391 has finished with status 0
Child exited normally with exit code 0
Parent process is dead now

```

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: Код программы к заданию №4

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5 #include <string.h>
6
7 #define RET_OK 0
8 #define RET_ERR_FORK 1
9 #define RET_ERR_PIPE 2
10
11 #define FORK_OK 0
12 #define FORK_ERR -1
13
14 #define INTERVAL 1
15 #define N_CHILDS 2
16
17 #define MSG1 "London is the capital of Great Britain\n"
18 #define LEN1 40
19 #define MSG2 "ABRA—kadabra\n"
20 #define LEN2 14
21 #define LENMAX 40
22

```

```

23 int main()
24 {
25     pid_t childpid1, childpid2, childpid;
26     int fd[2];
27
28     if (pipe(fd) == -1)
29     {
30         perror("Can't pipe\n");
31         return RET_ERR_PIPE;
32     }
33
34
35     if ((childpid1 = fork()) == FORK_ERR)
36     {
37         perror("Can't fork first child process.\n");
38         return RET_ERR_FORK;
39     }
40     else if (childpid1 == FORK_OK)
41     {
42         printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
43             getpid(), getppid(), getpgrp());
44
45         close(fd[0]);
46         write(fd[1], MSG1, strlen(MSG1) + 1);
47         printf("Message from first child was sent\n");
48
49         exit(RET_OK);
50     }
51
52     if ((childpid2 = fork()) == FORK_ERR)
53     {
54         perror("Can't fork second child process.\n");
55         return RET_ERR_FORK;
56     }
57     else if (childpid2 == FORK_OK)
58     {
59         printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
60             getpid(), getppid(), getpgrp());
61
62         close(fd[0]);
63         write(fd[1], MSG2, strlen(MSG2) + 1);
64         printf("Message from second child was sent\n");
65
66         exit(RET_OK);
67     }
68
69     sleep(INTERVAL);
70     printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n",
71         getpid(), getpgrp(), childpid1, childpid2);

```

```

72
73  int ch_status;
74  for (int i = 0; i < N_CHILDS; i++)
75  {
76      childpid = wait(&ch_status);
77      printf("Child with pid = %d has finished with status %d\n", childpid,
78             ch_status);
79
80      if (WIFEXITED(ch_status))
81          printf("Child exited normally with exit code %d\n", WEXITSTATUS(
82                 ch_status));
83      else if (WIFSIGNALED(ch_status))
84          printf("Child process ended with a non-intercepted signal number %d\n",
85                 WTERMSIG(ch_status));
86      else if (WIFSTOPPED(ch_status))
87          printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
88                 );
89  }
90
91  char message[LENMAX] = "\0";
92
93  printf("Reading messages from children.\n");
94  close(fd[1]);
95
96  if (read(fd[0], message, LEN1) < 0)
97      printf("No messages from first child.\n");
98  else
99      printf("Message from first child:\n%s", message);
100
101  if (read(fd[0], message, LEN2) < 0)
102      printf("No messages from second child.\n");
103  else
104      printf("Message from second child:\n%s", message);
105
106  printf("Parent process is dead now\n");
107  return RET_OK;
108 }

```

```

alena@DESKTOP-TJ9D65N:~/lab4$ ./task4
First child process: pid = 90, ppid = 89, pgrp = 89
Message from first child was sent
Second child process: pid = 91, ppid = 89, pgrp = 89
Message from second child was sent
Parent process: pid = 89, pgrp = 89, childpid1 = 90, childpid2 = 91
Child with pid = 90 has finished with status 0
Child exited normally with exit code 0
Child with pid = 91 has finished with status 0
Child exited normally with exit code 0
Reading messages from children.
Message from first child:
London is the capital of Great Britain
Message from second child:
ABRA-kadabra
Parent process is dead now

```

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

Задание №5

Предок и потомки аналогично п.4 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: Код программы к заданию №5

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5 #include <string.h>
6 #include <signal.h>
7
8 #define RET_OK 0
9 #define RET_ERR_FORK 1
10 #define RET_ERR_PIPE 2
11
12 #define FORK_OK 0
13 #define FORK_ERR -1
14
15 #define INTERVAL 2
16 #define N_CHILDS 2
17

```

```

18 #define MSG1 "London is the capital of Great Britain\n"
19 #define LEN1 40
20 #define MSG2 "ABRA—kadabra\n"
21 #define LEN2 14
22 #define LENMAX 40
23
24 short flag_writing_allowed = 0;
25
26 void allow_writing(int signal)
27 {
28     flag_writing_allowed = 1;
29     printf("\nSignal %d was caught. Writing is allowed now.\n", signal);
30 }
31
32 int main()
33 {
34     pid_t childpid1, childpid2, childpid;
35     int fd[2];
36
37     signal(SIGINT, allow_writing);
38     printf("Press \"CTRL+C\" to allow writing\n");
39     sleep(INTERVAL);
40
41     if (pipe(fd) == -1)
42     {
43         perror("Can't pipe\n");
44         return RET_ERR_PIPE;
45     }
46
47
48     if ((childpid1 = fork()) == FORK_ERR)
49     {
50         perror("Can't fork first child process.\n");
51         return RET_ERR_FORK;
52     }
53     else if (childpid1 == FORK_OK)
54     {
55         sleep(INTERVAL);
56         printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
57             getpid(), getppid(), getpgrp());
58
59         if (flag_writing_allowed)
60         {
61             close(fd[0]);
62             write(fd[1], MSG1, strlen(MSG1) + 1);
63             printf("Message from first child was sent\n");
64         }
65         else
66         {
67             printf("Writing to pipe for first child is not allowed\n");

```

```

68     }
69
70     exit(RET_OK);
71 }
72
73 if ((childpid2 = fork()) == FORK_ERR)
74 {
75     perror("Can't fork second child process.\n");
76     return RET_ERR_FORK;
77 }
78 else if (childpid2 == FORK_OK)
79 {
80     sleep(INTERVAL);
81     printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
82           getpid(), getppid(), getpgrp());
83
84     if (flag_writing_allowed)
85     {
86         close(fd[0]);
87         write(fd[1], MSG2, strlen(MSG2) + 1);
88         printf("Message from second child was sent\n");
89     }
90     else
91     {
92         printf("Writing to pipe for second child is not allowed\n");
93     }
94
95     exit(RET_OK);
96 }
97
98 printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n",
99       getpid(), getpgrp(), childpid1, childpid2);
100
101 sleep(INTERVAL);
102
103 int ch_status;
104 for (int i = 0; i < N_CHILDS; i++)
105 {
106     childpid = wait(&ch_status);
107     printf("Child with pid = %d has finished with status %d\n", childpid,
108           ch_status);
109
110     if (WIFEXITED(ch_status))
111         printf("Child exited normally with exit code %d\n", WEXITSTATUS(
112               ch_status));
113     else if (WIFSIGNALED(ch_status))
114         printf("Child process ended with a non-intercepted signal number %d\n",
115               WTERMSIG(ch_status));
116     else if (WIFSTOPPED(ch_status))

```

```

114     printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
        );
115 }
116
117 char message[LENMAX] = { 0 };
118
119 printf("Reading messages from children.\n");
120 close(fd[1]);
121
122 if (read(fd[0], message, LEN1) < 0)
123     printf("No messages from first child.\n");
124 else
125     printf("Message from first child:\n%s", message);
126
127 if (read(fd[0], message, LEN2) < 0)
128     printf("No messages from second child.\n");
129 else
130     printf("Message from second child:\n%s", message);
131
132 printf("Parent process is dead now\n");
133 return RET_OK;
134 }

```

```

alena@DESKTOP-TJ9D65N:~/lab4$ ./task5
Press "CTRL+C" to allow writing
^C
Signal 2 was caught. Writing is allowed now.
Parent process: pid = 87, pgrp = 87, childpid1 = 88, childpid2 = 89
First child process: pid = 88, ppid = 87, pgrp = 87
Message from first child was sent
Child with pid = 88 has finished with status 0
Second child process: pid = 89, ppid = 87, pgrp = 87
Child exited normally with exit code 0
Message from second child was sent
Child with pid = 89 has finished with status 0
Child exited normally with exit code 0
Reading messages from children.
Message from first child:
London is the capital of Great Britain
Message from second child:
ABRA-kadabra
Parent process is dead now

```

Рис. 5: Демонстрация работы программы при отправке сигнала (задание №5).

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task5
Press "CTRL+C" to allow writing
Parent process: pid = 125, pgrp = 125, childpid1 = 126, childpid2 = 127
First child process: pid = 126, ppid = 125, pgrp = 125
Writing to pipe for first child is not allowed
Second child process: pid = 127, ppid = 125, pgrp = 125
Child with pid = 126 has finished with status 0
Writing to pipe for second child is not allowed
Child exited normally with exit code 0
Child with pid = 127 has finished with status 0
Child exited normally with exit code 0
Reading messages from children.
Message from first child:
Message from second child:
Parent process is dead now
```

Рис. 6: Демонстрация работы программы при отсутствии отправки сигнала.