



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 Романов А.В.

Группа ИУ7-53Б

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 помков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе.

Листинг 1: Процессы-сироты

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define OK 0
5 #define FORK_FAILURE 1
6
7 #define N 2
8 #define SLP_INTV 2
9
10 int main()
11 {
12     int child[N];
13     int pid;
14     fprintf(stdout, "Parent process. PID: %d, GROUP: %d\n", getpid(), getpgrp());
15
16     for (size_t i = 0; i < N; i++)
17     {
18         pid = fork();
19         if (-1 == pid)
20         {
21             perror("Cant fork.");
22             return FORK_FAILURE;
23         }
24         else if (0 == pid)
25         {
26             sleep(SLP_INTV);
27             fprintf(stdout, "Child process #%d. PID: %d, PPID: %d, GROUP: %d\n", i
28                 + 1, getpid(), getppid(), getpgrp());
29             return OK;
30         } else
31         {
32             child[i] = pid;
33         }
34     }
35
36     fprintf(stdout, "Parent process. Children ID: %d, %d.\nParent process is
37         dead.\n", child[0], child[1]);
38     return OK;
39 }
```

```

alexey@alexey ~/reps/sem_05/os/sem_01/lab_04/src master gcc main_01.c
alexey@alexey ~/reps/sem_05/os/sem_01/lab_04/src master ./a.out
Parent process. PID: 9099, GROUP: 9099
Parent process. Children ID: 9100, 9101.
Parent process is dead.
alexey@alexey ~/reps/sem_05/os/sem_01/lab_04/src master Child process #1. PID: 9100, PPID: 1, GROUP: 9099
Child process #2. PID: 9101, PPID: 1, GROUP: 9099

```

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

Задание №2

Предок ждет завершения своих потомком, используя системный вызов `wait()`. Вывод соответствующих сообщений на экран.

Листинг 2: Вызов функции `wait()`

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5
6 #define OK 0
7 #define FORK_FAILURE 1
8
9 #define N 2
10 #define SLP_INTV 2
11
12 int main()
13 {
14     int child[N];
15     int pid;
16
17     fprintf(stdout, "Parent process. PID: %d, GROUP: %d\n", getpid(), getpgrp());
18
19     for (size_t i = 0; i < N; i++)
20     {
21         pid = fork();
22
23         if (-1 == pid)
24         {
25             perror("Cant fork.");
26             return FORK_FAILURE;
27         }
28         else if (0 == pid)
29         {
30             sleep(SLP_INTV);
31             fprintf(stdout, "Child process #%d. PID: %d, PPID: %d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
32

```

```

33     return OK;
34 } else
35 {
36     child[i] = pid;
37 }
38
39 }
40
41
42 for (size_t i = 0; i < N; i++)
43 {
44     int status, statval;
45
46     pid_t childpid = wait(&status);
47     fprintf(stdout, "Child process (PID %d) finished. Status: %d\n",
48             childpid, status);
49
50     if (WIFEXITED(statval))
51     {
52         fprintf(stdout, "Child process finished with code: %d\n", WEXITSTATUS(
53             statval));
54     }
55     else
56     {
57         fprintf(stdout, "Child process terminated abnormally\n");
58     }
59
60     fprintf(stdout, "Parent process. Children ID: %d, %d.\nParent process is
61         dead.\n", child[0], child[1]);
62
63     return OK;
64 }

```

```

alexey@alexey ~/reps/sem_05/os/sem_01/lab_04/src master • ./a.out
Parent process. PID: 14621, GROUP: 14621
Child process #1. PID: 14622, PPID: 14621, GROUP: 14621
Child process #2. PID: 14623, PPID: 14621, GROUP: 14621
Child process (PID 14622) finished. Status: 0
Child process finished with code: 0
Child process (PID 14623) finished. Status: 0
Child process finished with code: 0
Parent process. Children ID: 14622, 14623.
Parent process is dead.

```

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

Задание №3

Потомки переходят на выполнение других программ. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 3: Вызов функции `execvp()`

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5
6 #define OK 0
7 #define FORK_FAILURE 1
8
9 #define N 2
10 #define SLP_INTV 2
11
12 int main()
13 {
14     int child[N];
15     int pid;
16
17     fprintf(stdout, "Parent process. PID: %d, GROUP: %d\n", getpid(), getpgrp());
18
19     for (size_t i = 0; i < N; i++)
20     {
21         pid = fork();
22
23         if (-1 == pid)
24         {
25             perror("Cant fork.");
26             return FORK_FAILURE;
27         }
28         else if (0 == pid)
29         {
30             sleep(SLP_INTV);
31             fprintf(stdout, "Child process #%d. PID: %d, PPID: %d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
32
33             int rc = execvp(commands[i], commands[i], 0);
34
35             if (-1 == rc)
36             {
37                 perror("Cant exec.");
38                 return EXEC_FAILURE;
39             }
40
41             return OK;
42 }
```

```

43     } else
44     {
45         child[i] = pid;
46     }
47 }
48
49 for (size_t i = 0; i < N; i++)
50 {
51     int status, statval;
52
53     pid_t childpid = wait(&status);
54     fprintf(stdout, "Child process (PID %d) finished. Status: %d\n",
55             childpid, status);
56
57     if (WIFEXITED(statval))
58     {
59         fprintf(stdout, "Child process finished with code: %d\n", WEXITSTATUS(
60             statval));
61     }
62     else
63     {
64         fprintf(stdout, "Child process terminated abnormally\n");
65     }
66 }
67
68 fprintf(stdout, "Parent process. Children ID: %d, %d.\nParent process is
69     dead.\n", child[0], child[1]);
70
71 return OK;
72 }

```

```

alexey@alexey ~/repos/sem_05/os/sem_01/lab_04/src master • ./a.out
Parent process. PID: 20258, GROUP: 20258
Child process #1. PID: 20259, PPID: 20258, GROUP: 20258
Child process #2. PID: 20260, PPID: 20258, GROUP: 20258
a.out main_01.c main_02.c main_03.c
Child process (PID 20259) finished. Status: 0
Child process finished with code: 0
/home/alexey/repos/sem_05/os/sem_01/lab_04/src
Child process (PID 20260) finished. Status: 0
Child process finished with code: 0
Parent process. Children ID: 20259, 20260.
Parent process is dead.

```

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

Задание №4

Предок и потомки обмениваются сообщениями через неименованный программный канал. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 4: Использование pipe

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 #define OK 0
8 #define FORK_FAILURE 1
9 #define EXEC_FAILURE 2
10 #define PIPE_FAILURE 3
11
12 #define N 2
13 #define SLP_INTV 2
14 #define BUFSIZE 128
15
16 int main()
17 {
18     int child[N];
19     int fd[N];
20     int pid;
21
22     const char *const messages[N] = { "First message!\n", "Second message!\n"
23     };
24     char buffer[BUFSIZE] = { 0 };
25
26     if (-1 == pipe(fd))
27     {
28         perror("Cant pipe.");
29         return PIPE_FAILURE;
30     }
31
32     fprintf(stdout, "Parent process. PID: %d, GROUP: %d\n", getpid(), getpgrp
33     ());
34
35     for (size_t i = 0; i < N; i++)
36     {
37         pid = fork();
38
39         if (-1 == pid)
40         {
41             perror("Cant fork.");
42             return FORK_FAILURE;
43         }
44         else if (0 == pid)
```

```

43     {
44         close(fd[0]);
45         write(fd[1], messages[i], strlen(messages[i]));
46         fprintf(stdout, "Message #%d sent to parent!\n", i + 1);
47
48         return OK;
49     }
50     else
51     {
52         child[i] = pid;
53     }
54 }
55
56 for (size_t i = 0; i < N; i++)
57 {
58     int status, statval;
59
60     pid_t childpid = wait(&status);
61     fprintf(stdout, "Child process (PID %d) finished. Status: %d\n",
62             childpid, status);
63
64     if (WIFEXITED(statval))
65     {
66         fprintf(stdout, "Child process finished with code: %d\n", WEXITSTATUS(
67             statval));
68     }
69     else
70     {
71         fprintf(stdout, "Child process terminated abnormally\n");
72     }
73 }
74
75 close(fd[1]);
76 read(fd[0], buffer, BUFFSIZE);
77
78 fprintf(stdout, "Received messages:\n%s", buffer);
79 fprintf(stdout, "Parent process. Children ID: %d, %d.\nParent process is
80     dead.\n", child[0], child[1]);
81
82 return OK;
83 }

```



```

alexey@alexey > ~/reps/sem_05/os/sem_01/lab_04/src > master > ./a.out
Parent process. PID: 21997, GROUP: 21997
Message #1 sent to parent!
Message #2 sent to parent!
Child process (PID 21998) finished. Status: 0
Child process finished with code: 0
Child process (PID 21999) finished. Status: 0
Child process finished with code: 0
Received messages:
First message!
Second message!
Parent process. Children ID: 21998, 21999.
Parent process is dead.

```

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

Задание №5

Предок и потомки обмениваются сообщениями через неименованный программный канал. С помощью сигнала меняется ход выполнения программы. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 5: Использование сигналов

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 #define OK 0
8 #define FORK_FAILURE 1
9 #define EXEC_FAILURE 2
10 #define PIPE_FAILURE 3
11
12 #define GET 1
13 #define N 2
14 #define SLP_INTV 2
15 #define BUFSIZE 128
16
17 int mode = 0;
18
19 void dummy(int sigint) { }
20
21 void change_mode(int sigint)
22 {
23     mode = GET;
24 }
25
26 int main()

```

```

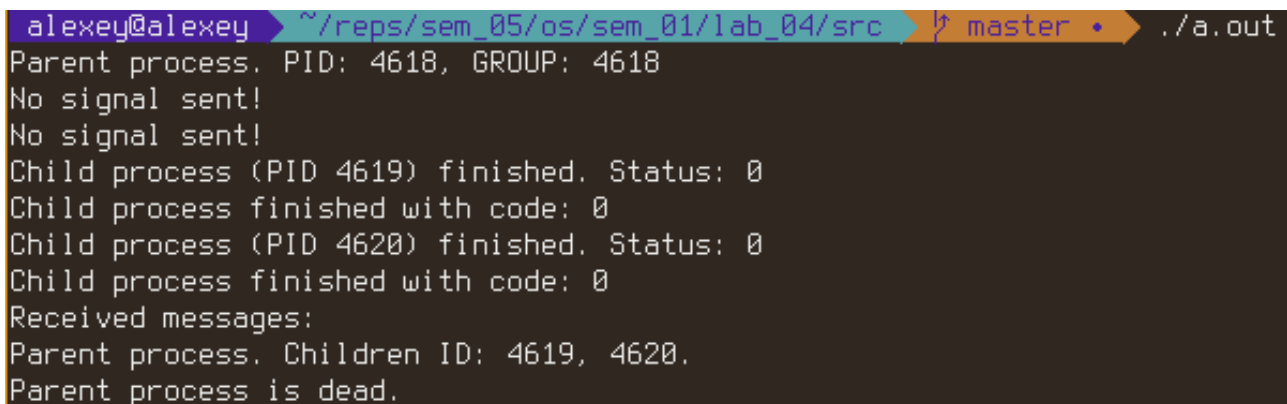
27 {
28     int child[N];
29     int fd[N];
30     int pid;
31
32     const char *const messages[N] = { "First message!\n", "Second message!\n"
33         };
34     char buffer[BUFFSIZE] = { 0 };
35
36     if (-1 == pipe(fd))
37     {
38         perror("Cant pipe.");
39         return PIPE_FAILURE;
40     }
41
42     fprintf(stdout, "Parent process. PID: %d, GROUP: %d\n", getpid(), getpgrp
43         ());
44     signal(SIGINT, dummy);
45
46     for (size_t i = 0; i < N; i++)
47     {
48         pid = fork();
49
50         if (-1 == pid)
51         {
52             perror("Cant fork.");
53             return FORK_FAILURE;
54         }
55         else if (0 == pid)
56         {
57             signal(SIGINT, change_mode);
58             sleep(SLP_INTV);
59
60             if (mode)
61             {
62                 close(fd[0]);
63                 write(fd[1], messages[i], strlen(messages[i]));
64                 fprintf(stdout, "Message #%d sent to parent!\n", i + 1);
65             }
66             else
67             {
68                 fprintf(stdout, "No signal sent!\n");
69             }
70
71             return OK;
72         }
73         else
74         {
75             child[i] = pid;

```

```

75     }
76 }
77
78 for (size_t i = 0; i < N; i++)
79 {
80     int status, statval;
81
82     pid_t childpid = wait(&status);
83     fprintf(stdout, "Child process (PID %d) finished. Status: %d\n",
84             childpid, status);
85
86     if (WIFEXITED(statval))
87     {
88         fprintf(stdout, "Child process finished with code: %d\n", WEXITSTATUS(
89             statval));
90     }
91     else
92     {
93         fprintf(stdout, "Child process terminated abnormally\n");
94     }
95 }
96
97 close(fd[1]);
98 read(fd[0], buffer, BUFFSIZE);
99
100 fprintf(stdout, "Received messages:\n%s", buffer);
101 fprintf(stdout, "Parent process. Children ID: %d, %d.\nParent process is
102     dead.\n", child[0], child[1]);
103
104 return OK;
105 }

```



```

alexey@alexey ~/reps/sem_05/os/sem_01/lab_04/src master • ./a.out
Parent process. PID: 4618, GROUP: 4618
No signal sent!
No signal sent!
Child process (PID 4619) finished. Status: 0
Child process finished with code: 0
Child process (PID 4620) finished. Status: 0
Child process finished with code: 0
Received messages:
Parent process. Children ID: 4619, 4620.
Parent process is dead.

```

Рис. 5: Демонстрация работы программы, сигнал не вызывается (задание №5).

```
alexey@alexey ~/reps/sem_05/os/sem_01/lab_04/src master • ./a.out
Parent process. PID: 4812, GROUP: 4812
^CMessage #1 sent to parent!
Message #2 sent to parent!
Child process (PID 4813) finished. Status: 0
Child process finished with code: 0
Child process (PID 4814) finished. Status: 0
Child process finished with code: 0
Received messages:
First message!
Second message!
Parent process. Children ID: 4813, 4814.
Parent process is dead.
```

Рис. 6: Демонстрация работы программы, сигнал вызывается (задание №5).