



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 Козлова И.В.

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

Листинг 1: Процессы-сироты

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 #define N 2
6 #define TIME_SLEEP 2
7
8 #define OK 0
9 #define ERR_FORK -1
10
11 #define FORK_FAILURE 1
12
13 int main()
14 {
15     int child[N];
16
17     printf("Parent process start! PID: %d, GROUP: %d\n", getpid(),
18           getpgrp());
19
20     for (int i = 0; i < N; i++)
21     {
22         int child_pid = fork();
23
24         if (child_pid == ERR_FORK)
25         {
26             perror("Can\'t fork()\n");
27             return FORK_FAILURE;
28         }
29         else if (!child_pid)
30         {
31             printf("BEFORE SLEEP Child %d! PID: %d, PPID: %d, GROUP: %d\n",
32                   i + 1, getpid(), getppid(), getpgrp());
33             sleep(TIME_SLEEP);
34             printf("AFTER SLEEP Child %d! PID: %d, PPID: %d, GROUP: %d\n",
35                   i + 1, getpid(), getppid(), getpgrp());
36             exit(OK);
37         }
38     }
39     else
40     {
41     }
```

```

37     child[i] = child_pid;
38 }
39 }
40
41 printf("Parent process finished! Children: %d, %d! \nParent: PID: %d,
      GROUP: %d\n ", child[0], child[1], getpid(), getpgrp());
42
43 return OK;
44 }

```

```

kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task1.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 21363, GROUP: 21363
Parent process finished! Children: 21364, 21365!
Parent: PID: 21363, GROUP: 21363
BEFORE SLEEP Child 1! PID: 21364, PPID: 21363, GROUP: 21363
BEFORE SLEEP Child 2! PID: 21365, PPID: 21363, GROUP: 21363
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ AFTER SLEEP Child 1!
  PID: 21364, PPID: 1481, GROUP: 21363
AFTER SLEEP Child 2! PID: 21365, PPID: 1481, GROUP: 21363

```

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

4	S	0	1103	1	0	80	0	-	63190	-	?	00:00:00	upowerd
4	S	121	1368	1	0	80	0	-	61659	-	?	00:00:00	colord
4	S	0	1441	895	0	80	0	-	41902	-	?	00:00:00	gdm-session-wor
4	S	1000	1481	1	0	80	0	-	4867	ep_pol	?	00:00:00	systemd
5	S	1000	1482	1481	0	80	0	-	42321	-	?	00:00:00	(sd-pam)
0	S	1000	1488	1481	8	69	-11	-	706448	poll_s	?	00:02:15	pulseaudio
0	S	1000	1490	1481	0	99	-	-	128012	poll_s	?	00:00:00	tracker-miner-f
0	S	1000	1494	1481	0	80	0	-	2225	ep_pol	?	00:00:00	dbus-daemon

Рис. 2: Процесс, который усыновляет процессы-сироты.

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 #include <stdlib.h>

```

```

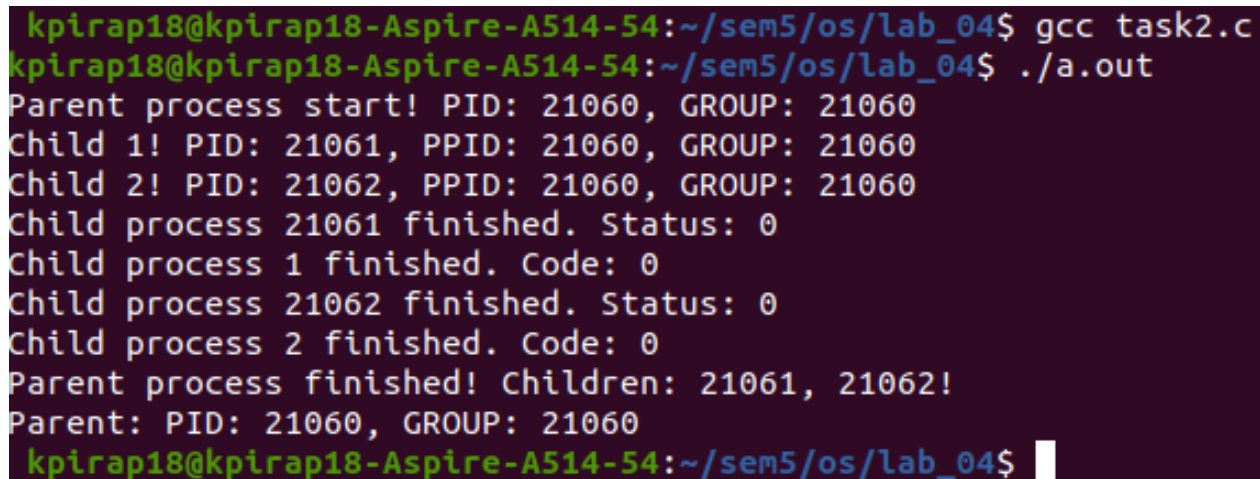
6
7 #define N 2
8 #define TIME_SLEEP 2
9
10 #define OK 0
11 #define ERR_FORK -1
12
13 #define FORK_FAILURE 1
14
15 int main()
16 {
17     int child[N];
18
19     printf("Parent process start! PID: %d, GROUP: %d\n", getpid(),
20         getpgrp());
21
22     for (int i = 0; i < N; i++)
23     {
24         int child_pid = fork();
25
26         if (child_pid == ERR_FORK)
27         {
28             perror("Can\'t fork()\n");
29             return FORK_FAILURE;
30         }
31         else if (!child_pid)
32         {
33             printf("Child %d! PID: %d, PPID: %d, GROUP: %d\n", i + 1,
34                 getpid(), getppid(), getpgrp());
35             exit(OK);
36         }
37         else
38         {
39             child[i] = child_pid;
40         }
41     }
42
43     for (int i = 0; i < N; i++)
44     {
45         int status;
46         int statval = 0;
47
48         pid_t child_pid = wait(&status);
49
50         printf("Child process %d finished. Status: %d\n", child_pid, status);
51
52         if (WIFEXITED(statval))

```

```

51 {
52     printf("Child process %d finished. Code: %d\n", i + 1,
           WEXITSTATUS(statval));
53 }
54 else if (WIFSIGNALED(statval))
55 {
56     printf("Child process %d finished from signal with code: %d\n", i
           + 1, WTERMSIG(statval));
57 }
58 else if (WIFSTOPPED(statval))
59 {
60     printf("Child process %d finished stopped. Number signal: %d\n",
           i + 1, WSTOPSIG(statval));
61 }
62 }
63
64 printf("Parent process finished! Children: %d, %d! \nParent: PID: %d,
        GROUP: %d\n ", child[0], child[1], getpid(), getpgrp());
65
66 return OK;
67 }

```



```

kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task2.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 21060, GROUP: 21060
Child 1! PID: 21061, PPID: 21060, GROUP: 21060
Child 2! PID: 21062, PPID: 21060, GROUP: 21060
Child process 21061 finished. Status: 0
Child process 1 finished. Code: 0
Child process 21062 finished. Status: 0
Child process 2 finished. Code: 0
Parent process finished! Children: 21061, 21062!
Parent: PID: 21060, GROUP: 21060
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$

```

Рис. 3: 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 `execlp()`

```

1 #include <stdio.h>

```

```

2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <stdlib.h>
6
7 #define N 2
8 #define TIME_SLEEP 2
9
10 #define OK 0
11 #define ERR_FORK -1
12 #define ERR_EXEC -1
13
14 #define FORK_FAILURE 1
15 #define EXEC_FAILURE 2
16
17 int main()
18 {
19     int child[N];
20     char *com[N] = {"/p1.exe", "/p2.exe"};
21
22     printf("Parent process start! PID: %d, GROUP: %d\n", getpid(),
23           getpgrp());
24
25     for (int i = 0; i < N; i++)
26     {
27         int child_pid = fork();
28
29         if (child_pid == ERR_FORK)
30         {
31             perror("Can't fork()\n");
32             return FORK_FAILURE;
33         }
34         else if (!child_pid)
35         {
36             printf("Child %d! PID: %d, PPID: %d, GROUP: %d\n", i + 1, getpid(),
37                   getppid(), getpgrp());
38             int rc = execlp(com[i], com[i], NULL);
39
40             if (rc == ERR_EXEC)
41             {
42                 perror("Can't exec");
43                 return EXEC_FAILURE;
44             }
45
46             exit(OK);
47         }
48     }
49     else
50     {
51

```

```

48     child[i] = child_pid;
49     }
50 }
51
52 for (int i = 0; i < N; i++)
53 {
54     int status;
55     int statval = 0;
56
57     pid_t child_pid = wait(&status);
58
59     printf("Child process %d finished. Status: %d\n", child_pid, status
60         );
61
62     if (WIFEXITED(statval))
63     {
64         printf("Child process %d finished. Code: %d\n", i + 1,
65             WEXITSTATUS(statval));
66     }
67     else if (WIFSIGNALED(statval))
68     {
69         printf("Child process %d finished from signal with code: %d\n", i
70             + 1, WTERMSIG(statval));
71     }
72     else if (WIFSTOPPED(statval))
73     {
74         printf("Child process %d finished stopped. Number signal: %d\n",
75             i + 1, WSTOPSIG(statval));
76     }
77 }
78
79 printf("Parent process finished! Children: %d, %d! \nParent: PID: %d,
80     GROUP: %d\n ", child[0], child[1], getpid(), getpgrp());
81
82 return OK;
83 }

```

Листинг 4: Код программы 1 (./p1.exe - исполняемый файл)

```

1 #include <stdio.h>
2
3 #define N 7
4
5 void sort_buble(int *mass, int n)
6 {
7     int no_swap = 0;
8     int tmp;
9
10    for (int i = n - 1; i >= 0; i--)

```

```

11 {
12     no_swap = 1;
13     for (int j = 0; j < i; j++)
14     {
15         if (mass[j] > mass[j + 1])
16         {
17             tmp = mass[j];
18             mass[j] = mass[j + 1];
19             mass[j + 1] = tmp;
20             no_swap = 0;
21         }
22     }
23     if (no_swap == 1)
24         break;
25 }
26 }
27
28 int main()
29 {
30     int mass[N] = {4, 9, 2, -1, 8, 3, 5};
31
32     printf("\n proc 1 (sort array) START\n");
33     printf("Arrat before: ");
34     for (int i = 0; i < N; i++)
35     {
36         printf("%d ", mass[i]);
37     }
38     printf("\n");
39
40     sort_buble(mass, N);
41
42     printf("Array after: ");
43     for (int i = 0; i < N; i++)
44     {
45         printf("%d ", mass[i]);
46     }
47     printf("\n proc 1 (sort array) END\n\n");
48     return 0;
49 }

```

Листинг 5: Код программы 2 (./p2.exe - исполняемый файл)

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #define N 16
5
6 void revstr(register char * str)
7 {

```



```

8  register int i, len = 0;
9  len = strlen(str);
10
11  for (i = 0; i <= len / 2; i++)
12  {
13      *(str + len - i) = *(str + i);
14      *(str + i) = *(str + len - i - 1);
15  }
16
17  for (i = len / 2; i <= len; i++)
18  {
19      *(str + i) = *(str + i + 1);
20  }
21
22  *(str + len) = '\0';
23 }
24
25 int main()
26 {
27     char str[] = "BMSTU IU7-52";
28
29     printf("\n proc 2 (reverse str) START\n");
30     printf("String before reverse: %s\n", str);
31
32     revstr(str);
33
34     printf("String after reverse: %s", str);
35     printf("\n proc 2 (reverse str) END\n\n");
36
37     return 0;
38 }

```

```

kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task3.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 21489, GROUP: 21489
Child 1! PID: 21490, PPID: 21489, GROUP: 21489
Child 2! PID: 21491, PPID: 21489, GROUP: 21489

    proc 1 (sort array) START
Array before: 4 9 2 -1 8 3 5
Array after: -1 2 3 4 5 8 9
    proc 1 (sort array) END

    proc 2 (reverse str) START
String before reverse: BMSTU IU7-52
String after reverse: 25-7UI UTSMB
    proc 2 (reverse str) END

Child process 21490 finished. Status: 0
Child process 1 finished. Code: 0
Child process 21491 finished. Status: 0
Child process 2 finished. Code: 0
Parent process finished! Children: 21490, 21491!
Parent: PID: 21489, GROUP: 21489

```

Рис. 4: 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 6: Использование pipe

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <string.h>
6
7 #define N 2
8 #define TIME_SLEEP 2
9 #define LEN 64
10
11 #define OK 0
12 #define ERR_FORK -1

```

```

13 #define ERR_EXEC -1
14 #define ERR_PIPE -1
15
16 #define FORK_FAILURE 1
17 #define EXEC_FAILURE 2
18 #define PIPE_FAILURE 3
19
20 int main()
21 {
22     int child[N];
23     int fd[N];
24     char text[LEN] = { 0 };
25     char *mes[N] = {"BMSTU IU7-52 Kozlova\n", "ABCDEFGH\n"};
26
27     if (pipe(fd) == ERR_PIPE)
28     {
29         perror("Can't pipe!");
30         return PIPE_FAILURE;
31     }
32
33     printf("Parent process start! PID: %d, GROUP: %d\n", getpid(),
34           getpgid());
35
36     for (int i = 0; i < N; i++)
37     {
38         int child_pid = fork();
39
40         if (child_pid == ERR_FORK)
41         {
42             perror("Can't fork()\n");
43             return ERR_FORK;
44         }
45         else if (!child_pid)
46         {
47             close(fd[0]);
48             write(fd[1], mes[i], strlen(mes[i]));
49             printf("Message %d sent to parent! %s", i + 1, mes[i]);
50
51             return OK;
52         }
53         else
54         {
55             child[i] = child_pid;
56         }
57     }
58
59     for (int i = 0; i < N; i++)
60     {

```

```

60     int status;
61     int statval = 0;
62
63     pid_t child_pid = wait(&status);
64
65     printf("Child process %d finished. Status: %d\n", child_pid, status
66           );
67
68     if (WIFEXITED(statval))
69     {
70         printf("Child process %d finished. Code: %d\n", i + 1,
71               WEXITSTATUS(statval));
72     }
73     else if (WIFSIGNALED(statval))
74     {
75         printf("Child process %d finished from signal with code: %d\n", i
76               + 1, WTERMSIG(statval));
77     }
78     else if (WIFSTOPPED(statval))
79     {
80         printf("Child process %d finished stopped. Number signal: %d\n",
81               i + 1, WSTOPSIG(statval));
82     }
83
84     printf("\nMessage receive :\n");
85     close(fd[1]);
86     read(fd[0], text, LEN);
87     printf("%s\n", text);
88
89     printf("Parent process finished! Children: %d, %d! \nParent: PID: %d,
90           GROUP: %d\n", child[0], child[1], getpid(), getpgrp());
91
92     return OK;
93 }

```

```

kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task4.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 21850, GROUP: 21850
Message 1 sent to parent! BMSTU IU7-52 Kozlova
Message 2 sent to parent! ABCDEFG
Child process 21851 finished. Status: 0
Child process 1 finished. Code: 0
Child process 21852 finished. Status: 0
Child process 2 finished. Code: 0

Message receive :
BMSTU IU7-52 Kozlova
ABCDEFGF

Parent process finished! Children: 21851, 21852!
Parent: PID: 21850, GROUP: 21850

```

Рис. 5: 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 7: Использование

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <string.h>
6 #include <stdbool.h>
7 #include <signal.h>
8
9 #define N 2
10 #define TIME_SLEEP 2
11 #define LEN 64
12
13 #define OK 0
14 #define ERR_FORK -1
15 #define ERR_EXEC -1
16 #define ERR_PIPE -1
17

```

```

18 #define FORK_FAILURE 1
19 #define EXEC_FAILURE 2
20 #define PIPE_FAILURE 3
21
22 _Bool flag = false;
23
24
25 void catch_sig(int sig_num)
26 {
27     flag = true;
28     printf("catch_sig: %d\n", sig_num);
29 }
30
31 int main()
32 {
33     int child[N];
34     int fd[N];
35     char text[LEN] = { 0 };
36     char *mes[N] = {"BMSTU IU7-52 Kozlova\n", "ABCDEFGH\n"};
37
38     if (pipe(fd) == ERR_PIPE)
39     {
40         perror("Can't pipe!");
41         return PIPE_FAILURE;
42     }
43
44     printf("Parent process start! PID: %d, GROUP: %d\n", getpid(),
45           getpgid());
46     signal(SIGINT, catch_sig);
47     sleep(2);
48
49     for (int i = 0; i < N; i++)
50     {
51         int child_pid = fork();
52
53         if (child_pid == ERR_FORK)
54         {
55             perror("Can't fork()\n");
56             return ERR_FORK;
57         }
58         else if (!child_pid)
59         {
60             if (flag)
61             {
62                 close(fd[0]);
63                 write(fd[1], mes[i], strlen(mes[i]));
64                 printf("Message %d sent to parent! %s", i + 1, mes[i]);

```

```

65     }
66
67     return OK;
68 }
69 else
70 {
71     child[i] = child_pid;
72 }
73 }
74
75 for (int i = 0; i < N; i++)
76 {
77     int status;
78     int statval = 0;
79
80     pid_t child_pid = wait(&status);
81
82     printf("Child process %d finished. Status: %d\n", child_pid, status
83         );
84
85     if (WIFEXITED(statval))
86     {
87         printf("Child process %d finished. Code: %d\n", i + 1,
88             WEXITSTATUS(statval));
89     }
90     else if (WIFSIGNALED(statval))
91     {
92         printf("Child process %d finished from signal with code: %d\n", i
93             + 1, WTERMSIG(statval));
94     }
95     else if (WIFSTOPPED(statval))
96     {
97         printf("Child process %d finished stopped. Number signal: %d\n",
98             i + 1, WSTOPSIG(statval));
99     }
100 }
101
102 printf("\nMessage receive :\n");
103 close(fd[1]);
104 read(fd[0], text, LEN);
105 printf("%s\n", text);
106
107 printf("Parent process finished! Children: %d, %d! \nParent: PID: %d,
108     GROUP: %d\n", child[0], child[1], getpid(), getpgrp());
109
110 return OK;
111 }

```

```

kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 18969, GROUP: 18969
Child process 18973 finished. Status: 0
Child process 1 finished. Code: 0
Child process 18974 finished. Status: 0
Child process 2 finished. Code: 0

Message receive :

Parent process finished! Children: 18973, 18974!
Parent: PID: 18969, GROUP: 18969
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ █

```

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

```

kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task5.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 21998, GROUP: 21998
^Ccatch_sig: 2
Message 1 sent to parent! BMSTU IU7-52
Message 2 sent to parent! ABCDEFG
Child process 21999 finished. Status: 0
Child process 1 finished. Code: 0
Child process 22000 finished. Status: 0
Child process 2 finished. Code: 0

Message receive :
BMSTU IU7-52
ABCDEFGF

Parent process finished! Children: 21999, 22000!
Parent: PID: 21998, GROUP: 21998

```

Рис. 7: Демонстрация работы программы, сигнал вызывается (задание №5).