



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»

---

## Отчёт

### по лабораторной работе №4

Название        «Оптимизация fork и exec»

---

Дисциплина    «Операционные системы»

---

Студент        ИУ7-55Б

---

(подпись, дата)

Бугаенко А.П.  
(Фамилия И.О.)

Преподаватель

---

(подпись, дата)

Рязанова Н.Ю.  
(Фамилия И.О.)

Москва, 2021

## Содержание

1	Теоретические сведения о системных вызовах fork и exec . . . . .	3
1.1	Системный вызов fork . . . . .	3
1.2	Системный вызов exec . . . . .	4
2	Листинг программ и результаты их работы . . . . .	6
2.1	Задание №1 . . . . .	6
2.2	Задание №2 . . . . .	7
2.3	Задание №3 . . . . .	9
2.4	Задание №4 . . . . .	11
2.5	Задание №5 . . . . .	13

# 1 Теоретические сведения о системных вызовах fork и exec

## 1.1 Системный вызов fork

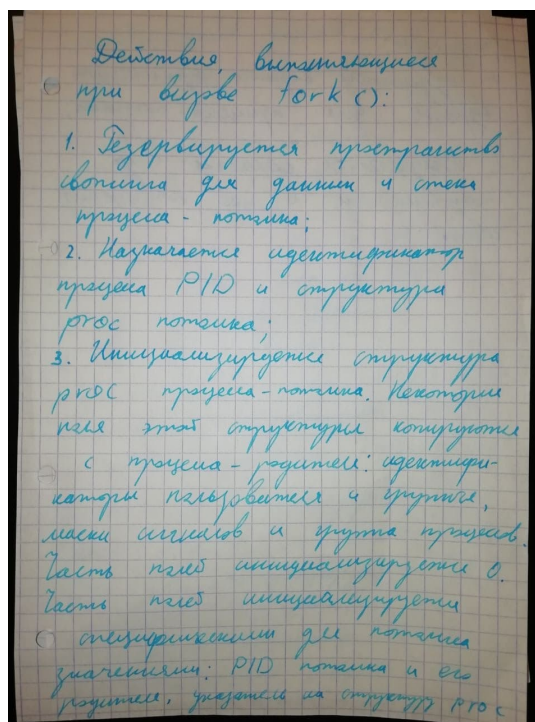


Рисунок 1.1 — Конспект fork - часть 1

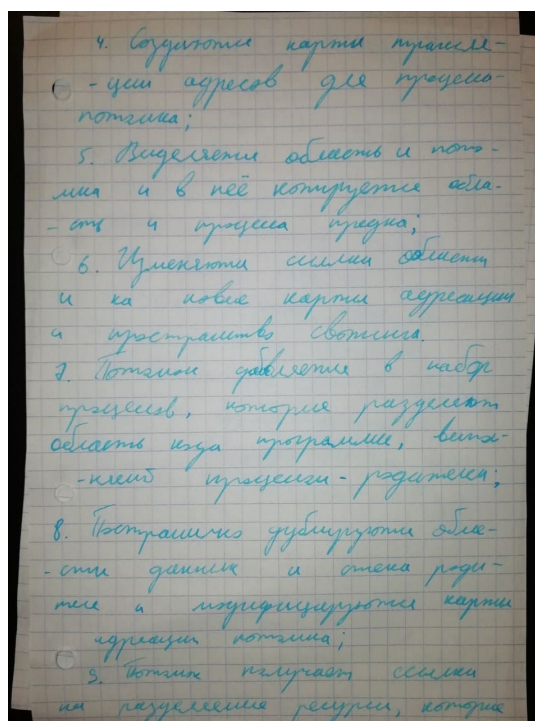


Рисунок 1.2 — Конспект fork - часть 2

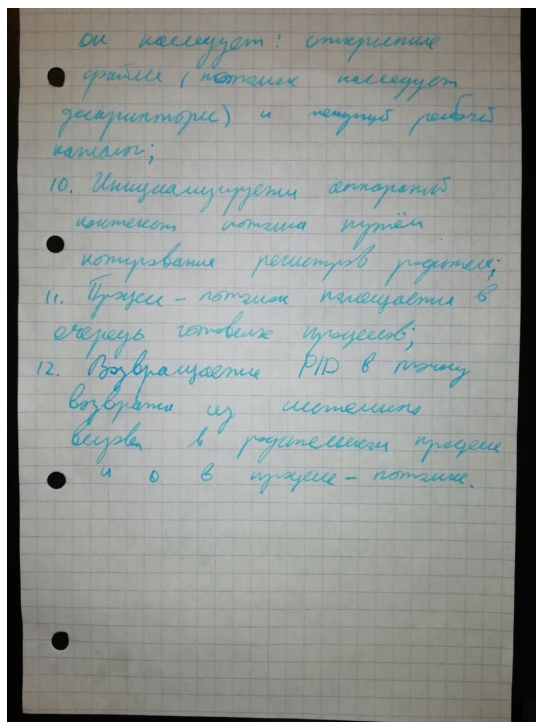


Рисунок 1.3 — Конспект fork - часть 3

## 1.2 Системный вызов exec

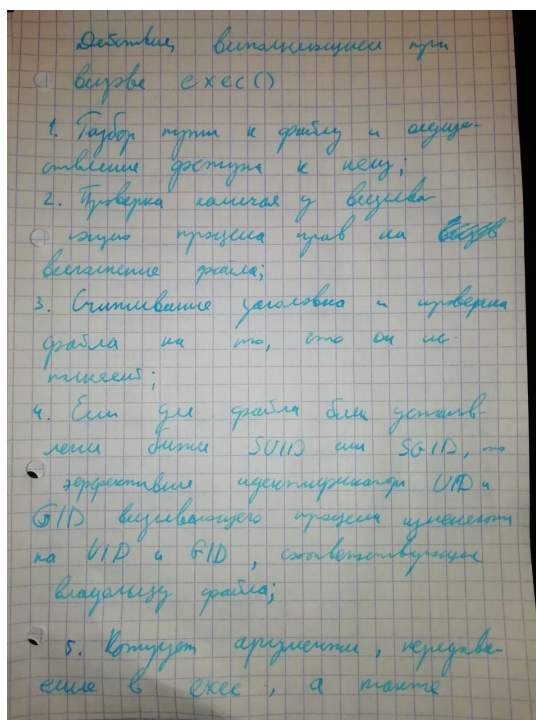


Рисунок 1.4 — Конспект exec - часть 1

переходимые среды в пространстве  
 зерна, после чего переходят  
 к переработке пространств  
 готово и уничтожить;

6. Визуализация пространства  
 Вспомогательная область данных  
 и текста;

7. Визуализация старого адресного  
 пространства и задания с тем  
 пространством Вспомогательная. Если же  
 процесс был создан при помощи Вспомогательной  
 функции Вспомогательная старого адресного  
 пространства радиальной функции;

8. Визуализация карты трансляции  
 адресов для новых данных, текста  
 и текста;

9. Уничтожение нового адресного  
 пространства. Если область текста  
 активна (каждый из двух процессов  
 уже выполнен эту программу), то

Рисунок 1.5 — Конспект ехес - часть 2

17. Визуализация аппарата  
 контроля. При этом Вспомогательная  
 функция Вспомогательная на 0, 4  
 указывает канал миграции  
 значения текста в новую программу.

Рисунок 1.6 — Конспект ехес - часть 3

## 2 Листинг программ и результаты их работы

### 2.1 Задание №1

Написать программу, запускающую не менее двух новых процессов системным вызовом `fork()`. В предке вывести собственный идентификатор (функция `getpid()`), идентификатор группы (функция `getpgrp()`) и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка (функция `getppid()`) и идентификатор группы. Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника.

Листинг 2.1 — Код к заданию №1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 #define SLEEP_TIME 2
6
7 int pid;
8 int child_pid_1;
9 int child_pid_2;
10
11 int main(void)
12 {
13     printf("Parent process | pid: %d | group: %d\\n", getpid(), getpgrp());
14
15     child_pid_1 = fork();
16     if (child_pid_1 == -1)
17     {
18         perror("Error: fork cannot be executed!");
19         return 1;
20     }
21     else if (child_pid_1 == 0)
22     {
23         sleep(SLEEP_TIME);
24         printf("Child 1 process | pid: %d | ppid: %d | group: %d\\n", getpid(),
25               getppid(), getpgrp());
26         exit(0);
27     }
28
29     child_pid_2 = fork();
30     if (child_pid_2 == -1)
31     {
32         perror("Error: fork cannot be executed!");
33         return 1;
34     }
35     else if (child_pid_2 == 0)
```

```

35     {
36         sleep(SLEEP_TIME);
37         printf("Child 2 process|pid: %d|ppid: %d|group: %d\n", getpid(),
               getppid(), getpgrp());
38         exit(0);
39     }
40     printf("Child PIDs of parent process|child 1: %d|child 2: %d\\n", child_pid_1,
           child_pid_1);
41     printf("End of parent process\\n");
42     return 0;
43 }

```

```

[andrew@andrew-virtualbox lab_4]$ ./task_1.exe
Parent process|pid: 9790|group: 9790
Child ids of parent process|child №1: 9791|child №2: 9791
End of parent process
[andrew@andrew-virtualbox lab_4]$
Child №1 process|pid: 9791|ppid: 1|group: 9790
Child №2 process|pid: 9792|ppid: 1|group: 9790

```

Рисунок 2.1 — Результат работы программы

## 2.2 Задание №2

Написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов `wait()`. Убедиться, что в этом случае идентификатор процесса потомка на 1 больше идентификатора процесса-предка.

Листинг 2.2 — Код к заданию №2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  #include <sys/types.h>
6  #include <sys/wait.h>
7
8  #define SLEEP_TIME 2
9
10 int pid;
11 int child_pid_1;
12 int child_pid_2;
13
14 int main(void)
15 {
16     printf("Parent process|pid: %d|group: %d\\n", getpid(), getpgrp());
17
18     child_pid_1 = fork();

```

```

19     if (child_pid_1 == -1)
20     {
21         perror("Error: fork cannot be executed!");
22         return 1;
23     }
24     else if (child_pid_1 == 0)
25     {
26         printf("Child 1 process | pid: %d | ppid: %d | group: %d\\n", getpid(),
27             getppid(), getpgrp());
28         exit(0);
29     }
30     child_pid_2 = fork();
31     if (child_pid_2 == -1)
32     {
33         perror("Error: fork cannot be executed!");
34         return 1;
35     }
36     else if (child_pid_2 == 0)
37     {
38         printf("Child 2 process | pid: %d | ppid: %d | group: %d\\n", getpid(),
39             getppid(), getpgrp());
40         exit(0);
41     }
42     int return_status[2];
43     pid_t child_pid[2];
44     for (int i = 0; i < 2; i++)
45     {
46         child_pid[i] = wait(&(return_status[i]));
47         printf("Child process finished | pid = %d | status = %d\\n", child_pid[i],
48             return_status[i]);
49         int status_value;
50         if (WIFEXITED(status_value))
51         {
52             printf("Child process exited succesfully with code %d\\n",
53                 WEXITSTATUS(status_value));
54         }
55         else
56         {
57             printf("Child process terminated abnormally\\n");
58         }
59     }
60     printf("Child pids of parent process | child 1: %d | child 2: %d\\n", child_pid_1,
61         child_pid_1);
62     printf("End of parent process\\n");

```



```

61     return 0;
62 }

```

```

[andrew@andrew-virtualbox lab_4]$ ./task_2.exe
Parent process|pid: 4792|group: 4792
Child №1 process|pid: 4793|ppid: 4792|group: 4792
Child process finished|pid = 4793|status = 0
Child process exited succesfully with code 0
Child №2 process|pid: 4794|ppid: 4792|group: 4792
Child process finished|pid = 4794|status = 0
Child process exited succesfully with code 0
Child pids of parent process|child №1: 4793|child №2: 4793
End of parent process

```

Рисунок 2.2 — Результат работы программы

### 2.3 Задание №3

Написать программу, в которой процесс-потомок вызывает системный вызов `exec()`, а процесс-предок ждет завершения процесса-потомка. Следует создать не менее двух потомков.

Листинг 2.3 — Код к заданию №3

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5  #include <unistd.h>
6
7  int pid;
8  int child_pid_1;
9  int child_pid_2;
10
11 int main(void)
12 {
13     printf("Parent process|pid: %d|group: %d\\n", getpid(), getpgrp());
14
15     child_pid_1 = fork();
16     if (child_pid_1 == -1)
17     {
18         perror("Error: fork cannot be executed!");
19         return 1;
20     }
21     else if (child_pid_1 == 0)
22     {
23         printf("Child 1 process|pid: %d|ppid: %d|group: %d\\n", getpid(),
24               getppid(), getpgrp());
25         int exec_result =
26             execlp("/media/sf_manjaro_shared/BMSTU/OS/labs/lab_4/test/test_sort_str.exe",
27                  "gadfasf", 0);

```

```

25     if (exec_result == -1)
26         perror("Error: execlp cannot be executed!");
27         return 2;
28     }
29
30     child_pid_2 = fork();
31     if (child_pid_2 == -1)
32     {
33         perror("Error: fork cannot be executed!");
34         return 1;
35     }
36     else if (child_pid_2 == 0)
37     {
38         printf("Child 2 process | pid: %d | ppid: %d | group: %d\\n", getpid(),
39             getppid(), getpgrp());
40         int exec_result =
41             execlp("/media/sf_manjaro_shared/BMSTU/OS/labs/lab_4/test/test_min_max.exe",
42                 "10", "5", "-1", "15", 0);
43         if (exec_result == -1)
44             perror("Error: execlp cannot be executed!");
45             return 2;
46     }
47
48     int return_status[2];
49     pid_t child_pid[2];
50     for (int i = 0; i < 2; i++)
51     {
52         child_pid[i] = wait(&(return_status[i]));
53         printf("Child process finished | pid = %d | status = %d\\n", child_pid[i],
54             return_status[i]);
55         int status_value;
56         if (WIFEXITED(status_value))
57         {
58             printf("Child process exited succesfully with code %d\\n",
59                 WEXITSTATUS(status_value));
60         }
61     }
62
63     printf("Child pids of parent process | child 1: %d | child 2: %d\\n", child_pid_1,
64         child_pid_1);
65     printf("End of parent process\\n");
66     return 0;
67 }

```

```

[andrew@andrew-virtualbox lab_4]$ ./task_3.exe
Parent process|pid: 6941|group: 6941
Child №1 process|pid: 6942|ppid: 6941|group: 6941
Child №2 process|pid: 6943|ppid: 6941|group: 6941
original: gadfasf|sorted: aadffgs
max: 15|min: 1
Child process finished|pid: 6942|status: 0
Child process finished|pid: 6943|status: 0
Child pids of parent process|child №1: 6942|child №2: 6942
End of parent process

```

Рисунок 2.3 — Результат работы программы

## 2.4 Задание №4

Написать программу, в которой предок и потомок обмениваются сообщением через программный канал.

Листинг 2.4 — Код к заданию №4

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <string.h>
5  #include <sys/wait.h>
6  #include <unistd.h>
7
8  int pid;
9  int child_pid_1;
10 int child_pid_2;
11
12 int main(void)
13 {
14     printf("Parent process|pid: %d|group: %d\\n", getpid(), getpgrp());
15
16     int message_pipe[2];
17     if (pipe(message_pipe) == -1)
18     {
19         perror("Error: pipe cannot be executed!");
20         return 1;
21     }
22
23     child_pid_1 = fork();
24     if (child_pid_1 == -1)
25     {
26         perror("Error: fork cannot be executed!");
27         return 1;
28     }
29     else if (child_pid_1 == 0)
30     {

```

```

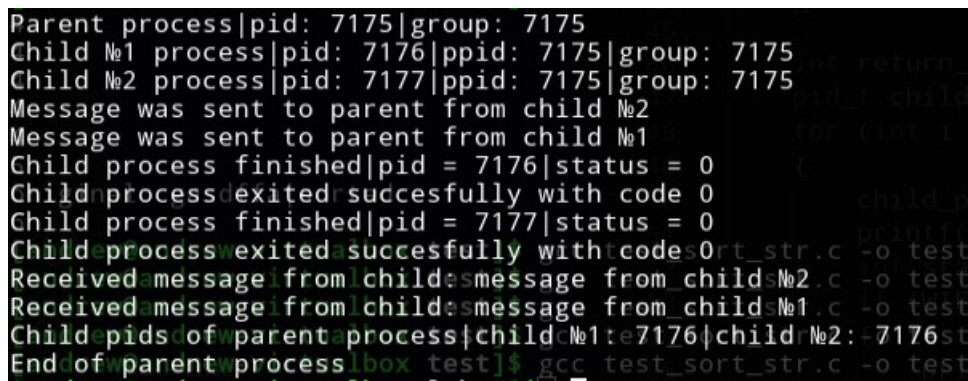
31     printf("Child 1 process |pid: %d|ppid: %d|group: %d\\n", getpid(),
           getppid(), getpgrp());
32     close(message_pipe[0]);
33     write(message_pipe[1], "message from child 1", strlen("message from child
           1") + 1);
34     printf("Message was sent to parent from child 1\\n");
35     return 0;
36 }
37
38 child_pid_2 = fork();
39 if (child_pid_2 == -1)
40 {
41     perror("Error: fork cannot be executed!");
42     return 1;
43 }
44 else if (child_pid_2 == 0)
45 {
46     printf("Child 2 process |pid: %d|ppid: %d|group: %d\\n", getpid(),
           getppid(), getpgrp());
47     close(message_pipe[0]);
48     write(message_pipe[1], "message from child 2", strlen("message from child
           2") + 1);
49     printf("Message was sent to parent from child 2\\n");
50     return 0;
51 }
52
53 int return_status[2];
54 pid_t child_pid[2];
55 for (int i = 0; i < 2; i++)
56 {
57     child_pid[i] = wait(&(return_status[i]));
58     printf("Child process finished |pid = %d|status = %d\\n", child_pid[i],
           return_status[i]);
59     int status_value;
60     if (WIFEXITED(status_value))
61     {
62         printf("Child process exited succesfully with code %d\\n",
           WEXITSTATUS(status_value));
63     }
64 }
65
66 char buff_1[24] = {0};
67 char buff_2[24] = {0};
68 close(message_pipe[1]);
69 read(message_pipe[0], buff_1, sizeof(buff_1));
70 read(message_pipe[0], buff_2, sizeof(buff_2));
71 printf("Received message from child: %s\\n", buff_1);

```

```

72     printf("Received message from child: %s\\n", buff_2);
73     printf("Child pids of parent process|child 1: %d|child 2: %d\\n", child_pid_1,
        child_pid_1);
74     printf("End of parent process\\n");
75     return 0;
76 }

```



```

Parent process|pid: 7175|group: 7175
Child №1 process|pid: 7176|ppid: 7175|group: 7175
Child №2 process|pid: 7177|ppid: 7175|group: 7175
Message was sent to parent from child №2
Message was sent to parent from child №1
Child process finished|pid = 7176|status = 0
Child process exited succesfully with code 0
Child process finished|pid = 7177|status = 0
Child process exited succesfully with code 0
Received message from child №2
Received message from child №1
Child pids of parent process|child №1: 7176|child №2: 7176
End of parent process

```

Рисунок 2.4 — Результат работы программы

## 2.5 Задание №5

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

Листинг 2.5 — Код к заданию №5

```

1  #include <signal.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7  #include <unistd.h>
8
9  int pid;
10 int child_pid_1;
11 int child_pid_2;
12
13 int signal_state = 0;
14
15 void ignore_signal(int signal) {printf("test");}
16 void writing(int signal) {printf("test"); signal_state = 1;}
17
18 int main(void)
19 {
20     printf("Parent process|pid: %d|group: %d\\n", getpid(), getppid());
21

```

```

22     int message_pipe[2];
23     if (pipe(message_pipe) == -1)
24     {
25         perror("Error: pipe cannot be executed!");
26         return 1;
27     }
28
29     signal(SIGINT, ignore_signal);
30     child_pid_1 = fork();
31     if (child_pid_1 == -1)
32     {
33         perror("Error: fork cannot be executed!");
34         return 1;
35     }
36     else if (child_pid_1 == 0)
37     {
38         signal(SIGINT, writing);
39         sleep(5);
40         if (signal_state == 1)
41         {
42             printf("Child 1 process | pid: %d | ppid: %d | group: %d\\n", getpid(),
43                    getppid(), getpgrp());
44             close(message_pipe[0]);
45             write(message_pipe[1], "message from child 1", strlen("message from
46                 child 1") + 1);
47             printf("Message was sent to parent from child 1\\n");
48         }
49         else
50         {
51             printf("No signal was sent\\n");
52         }
53         return 0;
54     }
55
56     child_pid_2 = fork();
57     if (child_pid_2 == -1)
58     {
59         perror("Error: fork cannot be executed!");
60         return 1;
61     }
62     else if (child_pid_2 == 0)
63     {
64         signal(SIGINT, writing);
65         sleep(5);
66         if (signal_state == 1)
67         {

```

```

66         printf("Child 2 process | pid: %d | ppid: %d | group: %d\\n", getpid(),
67             getppid(), getpgrp());
68         close(message_pipe[0]);
69         write(message_pipe[1], "message from child 2", strlen("message from
70             child 2") + 1);
71         printf("Message was sent to parent from child 2\\n");
72     }
73     else
74     {
75         printf("No signal was sent\\n");
76     }
77     return 0;
78 }
79
80 int return_status[2];
81 pid_t child_pid[2];
82 for (int i = 0; i < 2; i++)
83 {
84     child_pid[i] = wait(&(return_status[i]));
85     printf("Child process finished | pid = %d | status = %d\\n", child_pid[i],
86         return_status[i]);
87     int status_value;
88     if (WIFEXITED(status_value))
89     {
90         printf("Child process exited succesfully with code %d\\n",
91             WEXITSTATUS(status_value));
92     }
93 }
94
95 char buff_1[24] = {0};
96 char buff_2[24] = {0};
97 close(message_pipe[1]);
98 read(message_pipe[0], buff_1, sizeof(buff_1));
99 read(message_pipe[0], buff_2, sizeof(buff_2));
100 printf("Received message from child: %s\\n", buff_1);
101 printf("Received message from child: %s\\n", buff_2);
102 printf("Child pids of parent process | child 1: %d | child 2: %d\\n", child_pid_1,
103     child_pid_1);
104 printf("End of parent process\\n");
105 return 0;
106 }

```

```
Parent process|pid: 1698|group: 1698
No signal was sent
Child process finished|pid = 1700|status = 0
Child process exited succesfully with code 0
No signal was sent
Child process finished|pid = 1699|status = 0
Child process exited succesfully with code 0
Received message from child:
Received message from child:
Child pids of parent process|child №1: 1699|child №2: 1699
End of parent process
```

Рисунок 2.5 — Результат работы программы без сигнала

```
Parent process|pid: 1695|group: 1695
^CtestChild №2 process|pid: 1697|ppid: 1695|group: 1695
Message was sent to parent from child №2
testChild №1 process|pid: 1696|ppid: 1695|group: 1695
Message was sent to parent from child №1
testChild process finished|pid = 1696|status = 0
Child process exited succesfully with code 0
Child process finished|pid = 1697|status = 0
Child process exited succesfully with code 0
Received message from child: message from child №2
Received message from child: message from child №1
Child pids of parent process|child №1: 1696|child №2: 1696
End of parent process
```

Рисунок 2.6 — Результат работы программы с сигналом