



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 «ИУ, Информатика и системы управления»

КАФЕДРА «ИУ7, Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №4

по дисциплине

“Операционные системы”

Тема: Оптимизация fork

Студент: Андреев А.А.

Группа: ИУ7-54Б

Преподаватель: Рязанова Н.Ю.

Москва - 2021 г.

Оглавление

Оглавление	1
Задание 1	2
1. Постановка задачи	2
2. Листинг	2
3. Результат работы программы	4
Задание 2	5
1. Постановка задачи	5
2. Листинг	5
3. Результат работы программы	6
Задание 3	7
1. Постановка задачи	7
2. Листинг	7
3. Результат работы программы	9
Задание 4	10
1. Постановка задачи	10
2. Листинг	10
3. Результат работы программы	12
Задание 5	13
1. Постановка задачи	13
2. Листинг	13
3. Результат работы программы	16

Задание 1

1. Постановка задачи

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.

2. Листинг

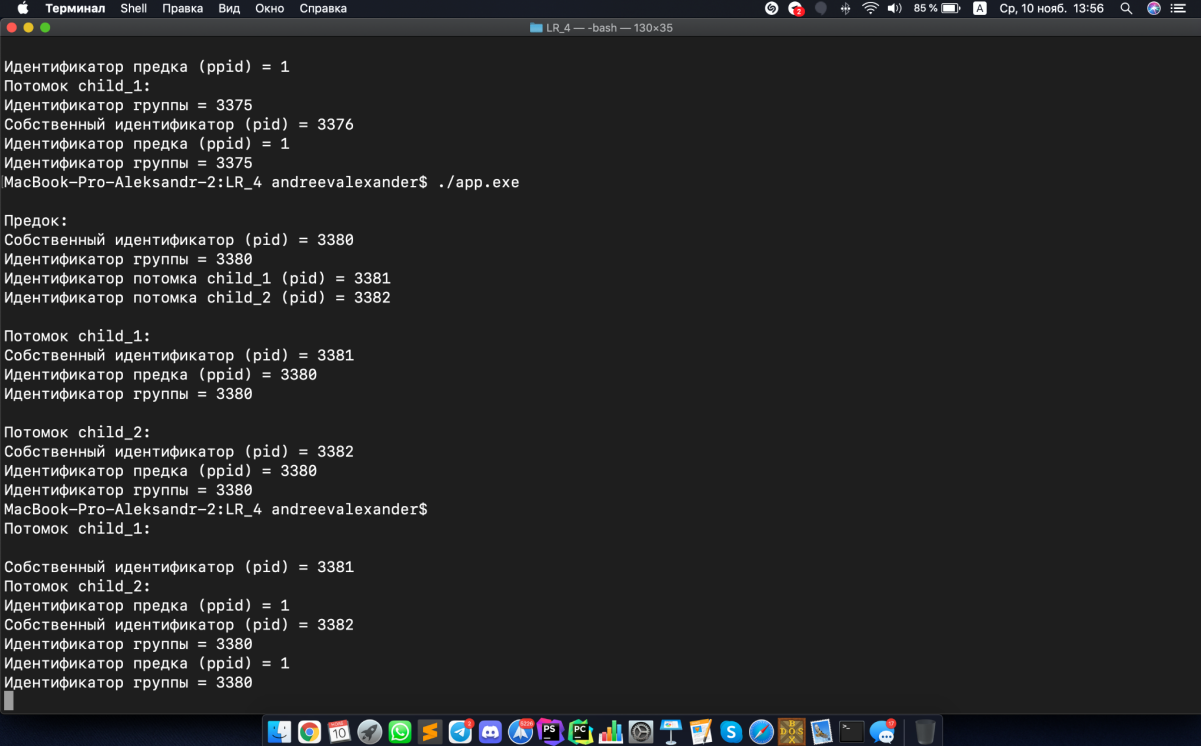
Листинг 1: Задача 1, Часть 1

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4.
5. int main()
6. {
7.     pid_t child_1, child_2;
8.
9.     if ((child_1 = fork()) == -1)
10.    {
11.        perror("Can't fork");
12.        exit(1);
13.    }
14.    else if (child_1 == 0)
15.    {
16.        printf("\nПотомок child_1:\n");
17.        printf("\nСобственный идентификатор (pid) = %d\n", getpid());
18.        printf("\nИдентификатор предка (ppid) = %d\n", getppid());
19.        printf("\nИдентификатор группы = %d\n", getpgrp());
20.        sleep(2);
21.        printf("\nПотомок child_1:\n");
22.        printf("\nСобственный идентификатор (pid) = %d\n", getpid());
23.        printf("\nИдентификатор предка (ppid) = %d\n", getppid());
24.        printf("\nИдентификатор группы = %d\n", getpgrp());
25.        exit(0);
26.    }
27.
28. }
```

Листинг 2: Задача 1, Часть 2

```
29.     if ((child_2 = fork()) == -1)
30.     {
31.         perror("Can't fork");
32.         exit(1);
33.     }
34.     else if (child_2 == 0)
35.     {
36.         printf("\nПотомок child_2:\n
37.             \nСобственный идентификатор (pid) = %d\
38.             \nИдентификатор предка (ppid) = %d\
39.             \nИдентификатор группы = %d\n",
40.             getpid(), getppid(), getpgrp());
41.         sleep(2);
42.         printf("\nПотомок child_2:\n
43.             \nСобственный идентификатор (pid) = %d\
44.             \nИдентификатор предка (ppid) = %d\
45.             \nИдентификатор группы = %d\n",
46.             getpid(), getppid(), getpgrp());
47.         exit(0);
48.     }
49.     printf("\nПредок:\n
50.         \nСобственный идентификатор (pid) = %d\
51.         \nИдентификатор группы = %d\
52.         \nИдентификатор потомка child_1 (pid) = %d\
53.         \nИдентификатор потомка child_2 (pid) = %d\n",
54.         getpid(), getpgrp(), child_1, child_2);
55.     return 0;
56. }
```

3. Результат работы программы



```
Идентификатор предка (ppid) = 1
Потомок child_1:
Идентификатор группы = 3375
Собственный идентификатор (pid) = 3376
Идентификатор предка (ppid) = 1
Идентификатор группы = 3375
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ ./app.exe

Предок:
Собственный идентификатор (pid) = 3380
Идентификатор группы = 3380
Идентификатор потомка child_1 (pid) = 3381
Идентификатор потомка child_2 (pid) = 3382

Потомок child_1:
Собственный идентификатор (pid) = 3381
Идентификатор предка (ppid) = 3380
Идентификатор группы = 3380

Потомок child_2:
Собственный идентификатор (pid) = 3382
Идентификатор предка (ppid) = 3380
Идентификатор группы = 3380
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$
Потомок child_1:

Собственный идентификатор (pid) = 3381
Потомок child_2:
Идентификатор предка (ppid) = 1
Собственный идентификатор (pid) = 3382
Идентификатор группы = 3380
Идентификатор предка (ppid) = 1
Идентификатор группы = 3380
```

Рисунок 1: Результат работы программы Задание 1

Задание 2

1. Постановка задачи

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.

2. Листинг

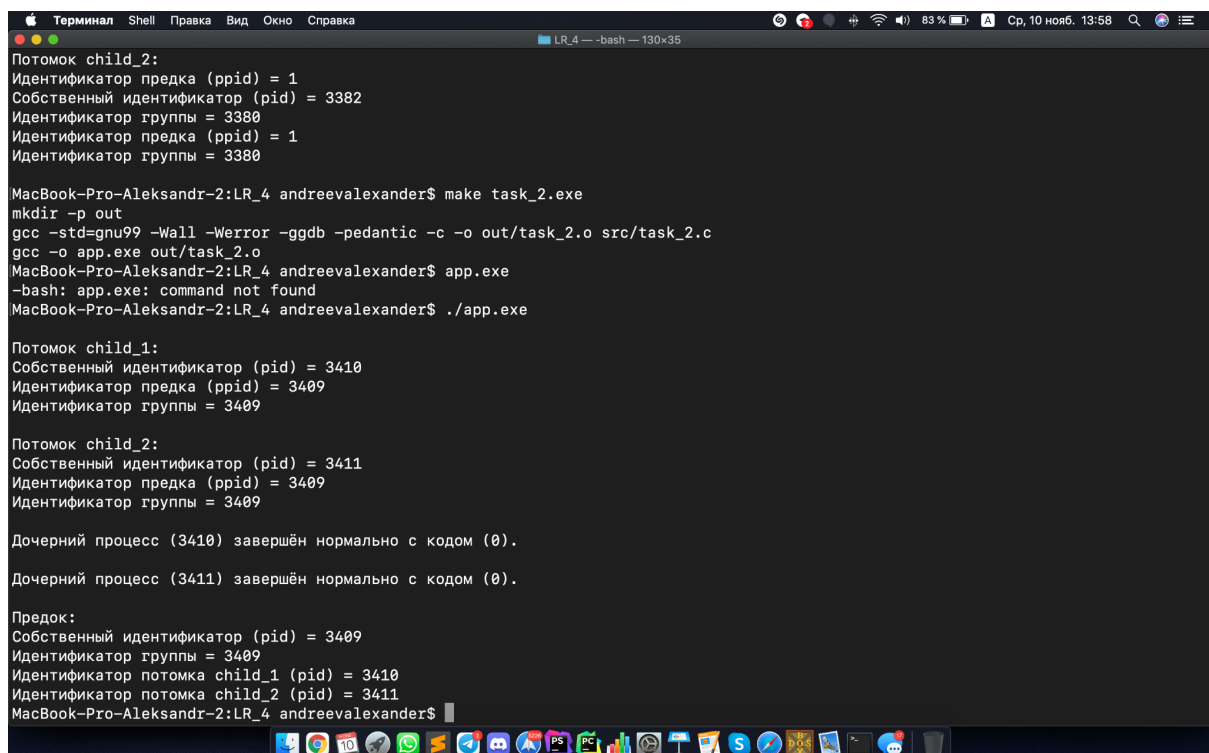
Листинг 3: Задача 2, Часть 1

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <sys/wait.h>
5.
6. int main()
7. {
8.     pid_t child_1, child_2;
9.     int status, val;
10.
11.     if ((child_1 = fork()) == -1)
12.     {
13.         perror("Fork failed");
14.         exit(1);
15.     }
16.
17.     else if (child_1 == 0)
18.     {
19.         printf("\nПотомок child_1:\n
20.             \nСобственный идентификатор (pid) = %d\n
21.             \nИдентификатор предка (ppid) = %d\n
22.             \nИдентификатор группы = %d\n",
23.             getpid(), getppid(), getpgrp());
24.         exit(0);
25.     }
26.
27.     if ((child_2 = fork()) == -1)
28.     {
29.         perror("Fork failed");
30.         exit(1);
31.     }
32.     else if (child_2 == 0)
33.     {
34.         printf("\nПотомок child_2:\n
35.             \nСобственный идентификатор (pid) = %d\n
36.             \nИдентификатор предка (ppid) = %d\n
37.             \nИдентификатор группы = %d\n",
38.             getpid(), getppid(), getpgrp());
```

Листинг 4: Задача 2, Часть 2

```
39.     exit(0);
40.     }
41.
42.     while ((val = wait(&status)) != -1) {
43.         if (WIFEXITED(status))
44.             printf("\nДочерний процесс (%d) завершён
нормально с кодом (%d).\n",
45.                 val, WEXITSTATUS(status));
46.         else if (WIFSIGNALED(status))
47.             printf("\nДочерний процесс (%d) завершён
неперехватываемым сигналом №(%d)\n", val,
48.                 WTERMSIG(status));
49.         else if (WIFSTOPPED(status))
50.             printf("\nДочерний процесс (%d) остановился,
номер сигнала: (%d)\n", val, WSTOPSIG(status));
51.     }
52.     printf("\nПредок:\n
53.         \nСобственный идентификатор (pid) = %d\n
54.         \nИдентификатор группы = %d\n
55.         \nИдентификатор потомка child_1 (pid) = %d\n
56.         \nИдентификатор потомка child_2 (pid) = %d\n",
57.         getpid(), getpgrp(), child_1, child_2);
58.     return 0;
59. }
```

3. Результат работы программы



```
Потомок child_2:
Идентификатор предка (ppid) = 1
Собственный идентификатор (pid) = 3382
Идентификатор группы = 3380
Идентификатор предка (ppid) = 1
Идентификатор группы = 3380

MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ make task_2.exe
mkdir -p out
gcc -std=gnu99 -Wall -Werror -ggdb -pedantic -c -o out/task_2.o src/task_2.c
gcc -o app.exe out/task_2.o
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ app.exe
-bash: app.exe: command not found
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ ./app.exe

Потомок child_1:
Собственный идентификатор (pid) = 3410
Идентификатор предка (ppid) = 3409
Идентификатор группы = 3409

Потомок child_2:
Собственный идентификатор (pid) = 3411
Идентификатор предка (ppid) = 3409
Идентификатор группы = 3409

Дочерний процесс (3410) завершён нормально с кодом (0).
Дочерний процесс (3411) завершён нормально с кодом (0).

Предок:
Собственный идентификатор (pid) = 3409
Идентификатор группы = 3409
Идентификатор потомка child_1 (pid) = 3410
Идентификатор потомка child_2 (pid) = 3411
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$
```

Задание 3

1. Постановка задачи

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

2. Листинг

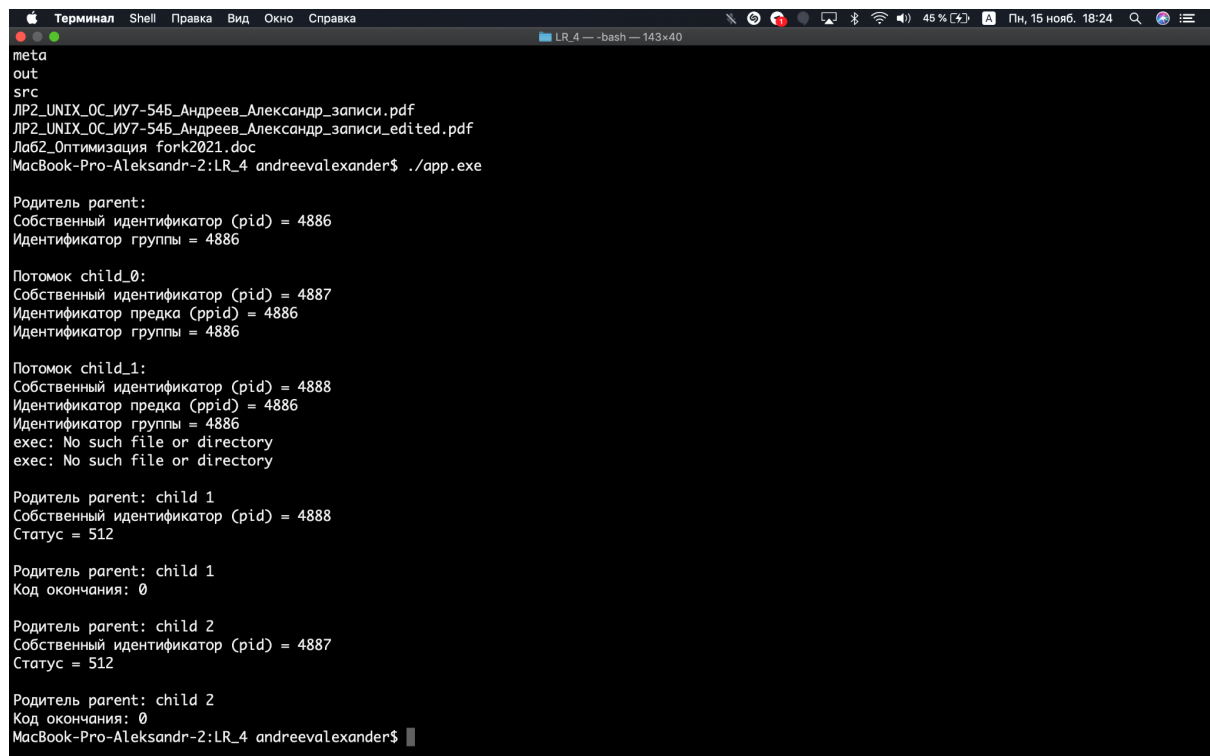
Листинг 5: Задача 3, Часть 1

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4.
5. #define NUMBER_OF_CHILDREN 2
6.
7. int main()
8. {
9.     char *p_exec[NUMBER_OF_CHILDREN] = {
10.         "meta/average/out/first_thread.o",
11.         "meta/factorial/out/second_thread.o"
12.     };
13.
14.     printf("\nРодитель parent:\n
15.         \nСобственный идентификатор (pid) = %d\n
16.         \nИдентификатор группы = %d\n",
17.         getpid(), getpgrp());
18.
19.     for (size_t i = 0; i < NUMBER_OF_CHILDREN; i++) {
20.         pid_t child;
21.
22.         if ((child = fork()) == -1)
23.         {
24.             perror("Can't fork");
25.             exit(1);
26.         }
27.
28.         else if (child == 0)
29.         {
30.             printf("\nПотомок child_%ld: \n
31.                 \nСобственный идентификатор (pid) = %d\n
32.                 \nИдентификатор предка (ppid) = %d\n
33.                 \nИдентификатор группы = %d\n",
34.                 i, getpid(), getppid(), getpgrp());
35.
36.             if (exec1(p_exec[i], NULL) == -1)
37.             {
```


Листинг 6: Задача 3, Часть 2

```
38.         perror("exec");
39.         exit(2);
40.     }
41.
42.     }
43. }
44.
45. for (size_t i = 0; i < NUMBER_OF_CHILDREN; i++) {
46.     int status = 0;
47.     int val = 0;
48.
49.     pid_t childpid = wait(&status);
50.
51.     printf("\nРодитель parent: child %ld\
52.           \nСобственный идентификатор (pid) = %d\
53.           \nСтатус = %d\n",
54.           i + 1, childpid, status);
55.
56.     if (WIFSIGNALED(val))
57.     {
58.         printf("\nРодитель parent: child %ld\
59.               \nКод окончания: %d\n", i + 1,
WTERMSIG(val));
60.     }
61.     else if (WIFEXITED(val))
62.     {
63.         printf("\nРодитель parent: child %ld\
64.               \nКод окончания: %d\n", i + 1,
WEXITSTATUS(val));
65.     }
66.     else if (WIFSTOPPED(val))
67.     {
68.         printf("\nРодитель parent: child %ld\
69.               \nКод окончания: %d\n", i + 1,
WSTOPSIG(val));
70.     }
71. }
72.
73. return 0;
74. }
```

3. Результат работы программы



```
meta
out
src
LP2_UNIX_OC_ИУ7-545_Андреев_Александр_записи.pdf
LP2_UNIX_OC_ИУ7-545_Андреев_Александр_записи_edited.pdf
Лаб2_Оптимизация_fork2021.doc
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ ./app.exe

Родитель parent:
Собственный идентификатор (pid) = 4886
Идентификатор группы = 4886

Потомок child_0:
Собственный идентификатор (pid) = 4887
Идентификатор предка (ppid) = 4886
Идентификатор группы = 4886

Потомок child_1:
Собственный идентификатор (pid) = 4888
Идентификатор предка (ppid) = 4886
Идентификатор группы = 4886
exec: No such file or directory
exec: No such file or directory

Родитель parent: child 1
Собственный идентификатор (pid) = 4888
Статус = 512

Родитель parent: child 1
Код окончания: 0

Родитель parent: child 2
Собственный идентификатор (pid) = 4887
Статус = 512

Родитель parent: child 2
Код окончания: 0
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$
```

Рисунок 3: Результат работы программы Задание 3

Задание 4

1. Постановка задачи

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

2. Листинг

Листинг 7: Задача 4, Часть 1

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <signal.h>
5. #include <sys/wait.h>
6.
7. int main()
8. {
9.     pid_t child_1, child_2;
10.    int status, val, fd[2];
11.
12.    if (pipe(fd) == -1)
13.    {
14.        perror("Pipe failed");
15.        exit(1);
16.    }
17.    if ((child_1 = fork()) == -1)
18.    {
19.        perror("Fork failed");
20.        exit(2);
21.    }
22.    else if (child_1 == 0)
23.    {
24.        printf("\nПотомок child_1:\n");
25.        printf("\nСобственный идентификатор (pid) = %d\n", pid);
26.        printf("\nИдентификатор предка (ppid) = %d\n", ppid);
27.        printf("\nИдентификатор группы = %d\n", pgid);
28.        getpid(), getppid(), getpgrp();
29.        char message[] = "kdjfskjfhdk sfhdsjhbncsdlkvjbf";
30.        close(fd[0]);
31.        write(fd[1], message, sizeof message - 1);
32.        printf("\nПотомок child_1 написал: %s\n",
33.            message);
34.        exit(0);
35.    }
36.    if ((child_2 = fork()) == -1)
37.    {
38.        perror("Fork failed");
39.        exit(3);
40.    }
```

Листинг 8: Задача 4, Часть 2

```
41.  else if (child_2 == 0)
42.  {
43.      printf("\nПотомок child_2:\n
44.              \nСобственный идентификатор (pid) = %d\n
45.              \nИдентификатор предка (ppid) = %d\n
46.              \nИдентификатор группы = %d\n",
47.              getpid(), getppid(), getpgrp());
48.      char message[] =
        "2094824093840928349023842093482093484234902834209384230948
        230498";
49.      close(fd[0]);
50.      write(fd[1], message, sizeof message - 1);
51.      printf("\nПотомок child_2 написал: %s\n",
        message);
52.      exit(0);
53.  }
54.
55.  while ((val = wait(&status)) != -1) {
56.      if (WIFEXITED(status))
57.          printf("\nДочерний процесс (%d) завершён
        нормально с кодом (%d).\n",
58.                  val, WEXITSTATUS(status));
59.      else if (WIFSIGNALED(status))
60.          printf("\nДочерний процесс (%d) завершён
        неперехватываемым сигналом №(%d)\n", val,
        WTERMSIG(status));
61.      else if (WIFSTOPPED(status))
62.          printf("\nДочерний процесс (%d) остановился,
        номер сигнала: (%d)\n", val, WSTOPSIG(status));
63.  }
64.
65.  printf("\nПредок:\n
66.          \nСобственный идентификатор (pid) = %d\n
67.          \nИдентификатор группы = %d\n
68.          \nИдентификатор потомка child_1 (pid) = %d\n
69.          \nИдентификатор потомка child_2 (pid) = %d\n",
70.          getpid(), getpgrp(), child_1, child_2);
71.
72.  close(fd[1]);
73.  char message[100];
74.  read(fd[0], message, sizeof message);
75.  printf("\nПредок прочитал: %s\n", message);
76.
77.  return 0;
78.  }
```

3. Результат работы программы

```
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ ./app.exe

Потомок child_1:
Собственный идентификатор (pid) = 4987
Идентификатор предка (ppid) = 4986
Идентификатор группы = 4986

Потомок child_1 написал: kdjfskjfhdksfhdsjhbncsdlkvjbf

Потомок child_2:
Собственный идентификатор (pid) = 4988
Идентификатор предка (ppid) = 4986
Идентификатор группы = 4986

Потомок child_2 написал: 2094824093840928349023842093482093484234902834209384230948230498

Дочерний процесс (4987) завершён нормально с кодом (0).
Дочерний процесс (4988) завершён нормально с кодом (0).

Предок:
Собственный идентификатор (pid) = 4986
Идентификатор группы = 4986
Идентификатор потомка child_1 (pid) = 4987
Идентификатор потомка child_2 (pid) = 4988

Предок прочитал: kdjfskjfhdksfhdsjhbncsdlkvjbf2094824093840928349023842093482093484234902834209384230948230498
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$
```

Рисунок 4: Результат работы программы Задание 4

Задание 5

1. Постановка задачи

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

2. Листинг

Листинг 9: Задача 5, Часть 1

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <signal.h>
5.
6. int child_1_write_flag = 0;
7. int child_2_write_flag = 0;
8.
9. void quit_signal_handler(int signum)
10. {
11.     child_1_write_flag = 1;
12.     child_2_write_flag = 1;
13. }
14.
15. int main()
16. {
17.     int val, status, fd[2];
18.     pid_t child_1, child_2;
19.
20.     if (pipe(fd) == -1)
21.     {
22.         perror("Pipe failed");
23.         exit(1);
24.     }
25.
26.     signal(SIGQUIT, quit_signal_handler);
27.
28.     if ((child_1 = fork()) == -1)
29.     {
30.         perror("Fork failed");
31.         exit(2);
32.     }
33.     else if (child_1 == 0)
34.     {
35.         printf("\nПотомок child_1:\n");
36.         printf("\nСобственный идентификатор (pid) = %d\n", pid);
37.         printf("\nИдентификатор предка (ppid) = %d\n", ppid);
38.         printf("\nИдентификатор группы = %d\n", grp);
39.         getpid(), getppid(), getpgrp();
40.         signal(SIGQUIT, quit_signal_handler);
41.         sleep(6);
42.     }
```

Листинг 10: Задача 5, Часть 2

```
43.  if (child_1_write_flag == 1)
44.      {
45.          char message[] = "(Сообщение от child_1)";
46.          close(fd[0]);
47.          write(fd[1], message, sizeof message - 1);
48.          printf("\nСигнал (Ctrl-\\) пришёл.");
49.          printf("\nПотомок child_1 написал: %s\n",
message);
50.      }
51.      else
52.      {
53.          printf("\nСигнал (Ctrl-\\) не пришёл.\n");
54.      }
55.      exit(0);
56.  }
57.
58.  if ((child_2 = fork()) == -1)
59.  {
60.      perror("Fork failed");
61.      exit(3);
62.  }
63.  else if (child_2 == 0)
64.  {
65.      printf("\nПотомок child_2:\n
66.          \nСобственный идентификатор (pid) = %d\n
67.          \nИдентификатор предка (ppid) = %d\n
68.          \nИдентификатор группы = %d\n",
69.          getpid(), getppid(), getpgrp());
70.      signal(SIGQUIT, quit_signal_handler);
71.      sleep(6);
72.      if (child_2_write_flag == 1)
73.      {
74.          char message[] = "(Сообщение от child_2)";
75.          close(fd[0]);
76.          write(fd[1], message, sizeof message - 1);
77.          printf("\nСигнал (Ctrl-\\) пришёл.");
78.          printf("\nПотомок child_2 написал: %s\n",
message);
79.      }
80.      else
81.      {
82.          printf("\nСигнал (Ctrl-\\) не пришёл.\n");
83.      }
84.      exit(0);
85.  }
86.
87.  while ((val = wait(&status)) != -1) {
88.      if (WIFEXITED(status))
89.          printf("\nДочерний процесс (%d) завершён
нормально с кодом (%d).\n",
90.          val, WEXITSTATUS(status));
91.
```


Листинг 11: Задача 5, Часть 3

```
92.     else if (WIFSIGNALED(status))
93.         printf("\nДочерний процесс (%d) завершён\n", val,
WTERMSIG(status));
94.     else if (WIFSTOPPED(status))
95.         printf("\nДочерний процесс (%d) остановился,\n", val, WSTOPSIG(status));
96.     }
97.
98.     printf("\nПредок:\n");
99.     printf("\nСобственный идентификатор (pid) = %d\n", pid);
100.    printf("\nИдентификатор группы = %d\n", pgrp);
101.    printf("\nИдентификатор потомка child_1 (pid) = %d\n", child_1_pid);
102.    printf("\nИдентификатор потомка child_2 (pid) = %d\n", child_2_pid);
103.    printf("getpid(), getpgrp(), child_1, child_2);\n");
104.
105.    close(fd[1]);
106.    char message[100];
107.    read(fd[0], message, sizeof message);
108.    printf("\nПредок прочитал: %s\n", message);
109.
110.    return 0;
111. }
```

3. Результат работы программы

```
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ ./app.exe

Потомок child_1:
Собственный идентификатор (pid) = 3511
Идентификатор предка (ppid) = 3510
Идентификатор группы = 3510

Потомок child_2:
Собственный идентификатор (pid) = 3512
Идентификатор предка (ppid) = 3510
Идентификатор группы = 3510

Сигнал (Ctrl-\) не пришёл.
Сигнал (Ctrl-\) не пришёл.

Дочерний процесс (3512) завершён нормально с кодом (0).

Дочерний процесс (3511) завершён нормально с кодом (0).

Предок:
Собственный идентификатор (pid) = 3510
Идентификатор группы = 3510
Идентификатор потомка child_1 (pid) = 3511
Идентификатор потомка child_2 (pid) = 3512

Предок прочитал:
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$
```

Рисунок 5: Результат работы программы Задание 5: Сигнал не получен

```
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$ ./app.exe
```

```
Потомок child_2:
```

```
Потомок child_1:
```

```
Собственный идентификатор (pid) = 3519
```

```
Собственный идентификатор (pid) = 3518
```

```
Идентификатор предка (ppid) = 3517
```

```
Идентификатор предка (ppid) = 3517
```

```
Идентификатор группы = 3517
```

```
Идентификатор группы = 3517
```

```
^\
```

```
Сигнал (Ctrl-\) пришёл.
```

```
Потомок child_2 написал: (Сообщение от child_2)
```

```
Сигнал (Ctrl-\) пришёл.
```

```
Потомок child_1 написал: (Сообщение от child_1)
```

```
Дочерний процесс (3519) завершён нормально с кодом (0).
```

```
Дочерний процесс (3518) завершён нормально с кодом (0).
```

```
Предок:
```

```
Собственный идентификатор (pid) = 3517
```

```
Идентификатор группы = 3517
```

```
Идентификатор потомка child_1 (pid) = 3518
```

```
Идентификатор потомка child_2 (pid) = 3519
```

```
Предок прочитал: (Сообщение от child_2)(Сообщение от child_1)
```

```
MacBook-Pro-Aleksandr-2:LR_4 andreevalexander$
```

Рисунок 6: Результат работы программы Задание 5: Сигнал был получен