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.

Москва — 2021 г.

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

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define RET_OK 0
#define RET_ERR_FORK 1

#define FORK_OK 0
#define FORK_ERR -1

#define INTERVAL 1

int main()
{
  pid_t childpid1, childpid2;
  if ((childpid1 = fork()) == FORK_ERR)
  {
    perror("Can't fork first child process.\n");
    return RET_ERR_FORK;
  }
  else if (childpid1 == FORK_OK)
  {
    printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
    getpid(), getppid(), getpgrp());

    sleep(INTERVAL);
    printf("First child process (has become an orphan): pid = %d, ppid = %d,
        pgrp = %d\n",
    getpid(), getppid(), getpgrp());

    printf("First child process is dead now\n");

    exit(RET_OK);
  }


  if ((childpid2 = fork()) == FORK_ERR)
  {
    perror("Can't fork second child process.\n");
    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
              , pgrp = %d\n",
48        getpid(), getppid(), getpgrp());
49
50        printf("Second child process is dead now\n");
51        exit(RET_OK);
52    }
53
54    printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
          d\n",
55    getpid(), getpgrp(), childpid1, childpid2);
56    printf("Parent process is dead now\n");
57    return RET_OK;
58
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 потомком, 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
```

```c
#define FORK_OK 0
#define FORK_ERR -1

#define INTERVAL 1

int main()
{
  pid_t childpid1, childpid2, childpid;
  if ((childpid1 = fork()) == FORK_ERR)
  {
    perror("Can't fork first child process.\n");
    return RET_ERR_FORK;
  }
  else if (childpid1 == FORK_OK)
  {
    printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
    getpid(), getppid(), getpgrp());

    exit(RET_OK);
  }

  if ((childpid2 = fork()) == FORK_ERR)
  {
    perror("Can't fork second child process.\n");
    return RET_ERR_FORK;
  }
  else if (childpid2 == FORK_OK)
  {
    printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
    getpid(), getppid(), getpgrp());

    exit(RET_OK);
  }

  sleep(INTERVAL);
  printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
     d\n",
  getpid(), getpgrp(), childpid1, childpid2);

  int ch_status;
  for (int i = 0; i < 2; i++)
  {
    childpid = wait(&ch_status);
    printf("Child with pid = %d has finished with status %d\n", childpid,
       ch_status);

    if (WIFEXITED(ch_status))
    printf("Child exited normally with exit code %d\n", WEXITSTATUS(
       ch_status));
    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. Предок 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     printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
43     getpid(), getppid(), getpgrp());
44     if (execlp("cat", "cat", "for_cat.txt", NULL) < 0)
45     {
46       perror("Can't execlp from second child.\n");
47       exit(RET_CANT_EXECLP);
48     }
49
50     exit(RET_OK);
51   }
52
53   sleep(INTERVAL);
54   printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
       d\n",
55   getpid(), getpgrp(), childpid1, childpid2);
56
57   int ch_status;
58   for (int i = 0; i < 2; i++)
59   {
60     childpid = wait(&ch_status);
61     printf("Child with pid = %d has finished with status %d\n", childpid,
         ch_status);
62
63     if (WIFEXITED(ch_status))
```

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

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 }
```

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task3
First child process: pid = 179, ppid = 178, pgrp = 178
Second child process: pid = 180, ppid = 178, pgrp = 178
This is program called from first child
I can count sum: 2 + 3 = 5
This is text in file for_cat.txt
Parent process: pid = 178, pgrp = 178, childpid1 = 179, childpid2 = 180
Child with pid = 179 has finished with status 0
Child exited normally with exit code 0
Child with pid = 180 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 5: Код программы к заданию №4

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <string.h>

#define RET_OK 0
#define RET_ERR_FORK 1
#define RET_ERR_PIPE 2

#define FORK_OK 0
#define FORK_ERR -1

#define INTERVAL 1
#define N_CHILDS 2

#define MSG1 "ABCDEFG\n"
#define LEN1 9
#define MSG2 "ZXY\n"
#define LEN2 5
#define LENMAX 9

```

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

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task4
First child process: pid = 197, ppid = 196, pgrp = 196
Message from first child was sent
Second child process: pid = 198, ppid = 196, pgrp = 196
Message from second child was sent
Parent process: pid = 196, pgrp = 196, childpid1 = 197, childpid2 = 198
Child with pid = 197 has finished with status 0
Child exited normally with exit code 0
Child with pid = 198 has finished with status 0
Child exited normally with exit code 0
Reading messages from children.
Message from first child:
ABCDEFG
Message from second child:
ZXY
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 6: Код программы к заданию №5

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <string.h>
#include <signal.h>

#define RET_OK 0
#define RET_ERR_FORK 1
#define RET_ERR_PIPE 2

#define FORK_OK 0
#define FORK_ERR -1

#define INTERVAL 1
#define N_CHILDS 2

#define MSG1 "ABCDEFG\n"
```

```
19 #define LEN1 9
20 #define MSG2 "ZXY\n"
21 #define LEN2 5
22 #define LENMAX 9
23
24 short flag_writing_allowed = 0;
25
26 void allow_writing(int signal)
27 {
28   flag_writing_allowed = 1;
29 }
30
31 int main()
32 {
33   pid_t childpid1, childpid2, childpid;
34   int fd[2];
35
36   // signal handler assignment
37   signal(SIGUSR1, allow_writing);
38
39   if (pipe(fd) == -1)
40   {
41     perror("Can't pipe\n");
42     return RET_ERR_PIPE;
43   }
44
45
46   if ((childpid1 = fork()) == FORK_ERR)
47   {
48     perror("Can't fork first child process.\n");
49     return RET_ERR_FORK;
50   }
51   else if (childpid1 == FORK_OK)
52   {
53     sleep(INTERVAL);
54     printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
55     getpid(), getppid(), getpgrp());
56
57     if (flag_writing_allowed)
58     {
59       close(fd[0]);
60       write(fd[1], MSG1, strlen(MSG1) + 1);
61       printf("Message from first child was sent\n");
62     }
63     else
64     {
65       printf("Writing to pipe for first child is not allowed\n");
66     }
67
68     exit(RET_OK);
```

```
69    }
70
71    if ((childpid2 = fork()) == FORK_ERR)
72    {
73      perror("Can't fork second child process.\n");
74      return RET_ERR_FORK;
75    }
76    else if (childpid2 == FORK_OK)
77    {
78      sleep(INTERVAL);
79      printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
80      getpid(), getppid(), getpgrp());
81
82      if (flag_writing_allowed)
83      {
84        close(fd[0]);
85        write(fd[1], MSG2, strlen(MSG2) + 1);
86        printf("Message from second child was sent\n");
87      }
88      else
89      {
90        printf("Writing to pipe for second child is not allowed\n");
91      }
92
93      exit(RET_OK);
94    }
95
96    printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
         d\n",
97    getpid(), getpgrp(), childpid1, childpid2);
98
99    // send signals
100   kill(childpid1, SIGUSR1);
101   kill(childpid2, SIGUSR1);
102
103   sleep(INTERVAL);
104
105   int ch_status;
106   for (int i = 0; i < N_CHILDS; i++)
107   {
108     childpid = wait(&ch_status);
109     printf("Child with pid = %d has finished with status %d\n", childpid,
          ch_status);
110
111     if (WIFEXITED(ch_status))
112     printf("Child exited normally with exit code %d\n", WEXITSTATUS(
          ch_status));
113     else if (WIFSIGNALED(ch_status))
114     printf("Child process ended with a non-intercepted signal number %d\n",
          WTERMSIG(ch_status));
```

```
115       else if (WIFSTOPPED(ch_status))
116       printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
              );
117   }
118
119   char message[LENMAX] = { 0 };
120
121   printf("Reading messages from children.\n");
122   close(fd[1]);
123
124   if (read(fd[0], message, LEN1) < 0)
125   printf("No messages from first child.\n");
126   else
127   printf("Message from first child:\n%s", message);
128
129   if (read(fd[0], message, LEN2) < 0)
130   printf("No messages from second child.\n");
131   else
132   printf("Message from second child:\n%s", message);
133
134   printf("Parent process is dead now\n");
135   return RET_OK;
136 }
```

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task5
Parent process: pid = 309, pgrp = 309, childpid1 = 310, childpid2 = 311
First child process: pid = 310, ppid = 309, pgrp = 309
Second child process: pid = 311, ppid = 309, pgrp = 309
Message from first child was sent
Message from second child was sent
Child with pid = 310 has finished with status 0
Child exited normally with exit code 0
Child with pid = 311 has finished with status 0
Child exited normally with exit code 0
Reading messages from children.
Message from first child:
ABCDEFG
Message from second child:
ZXY
Parent process is dead now
```

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

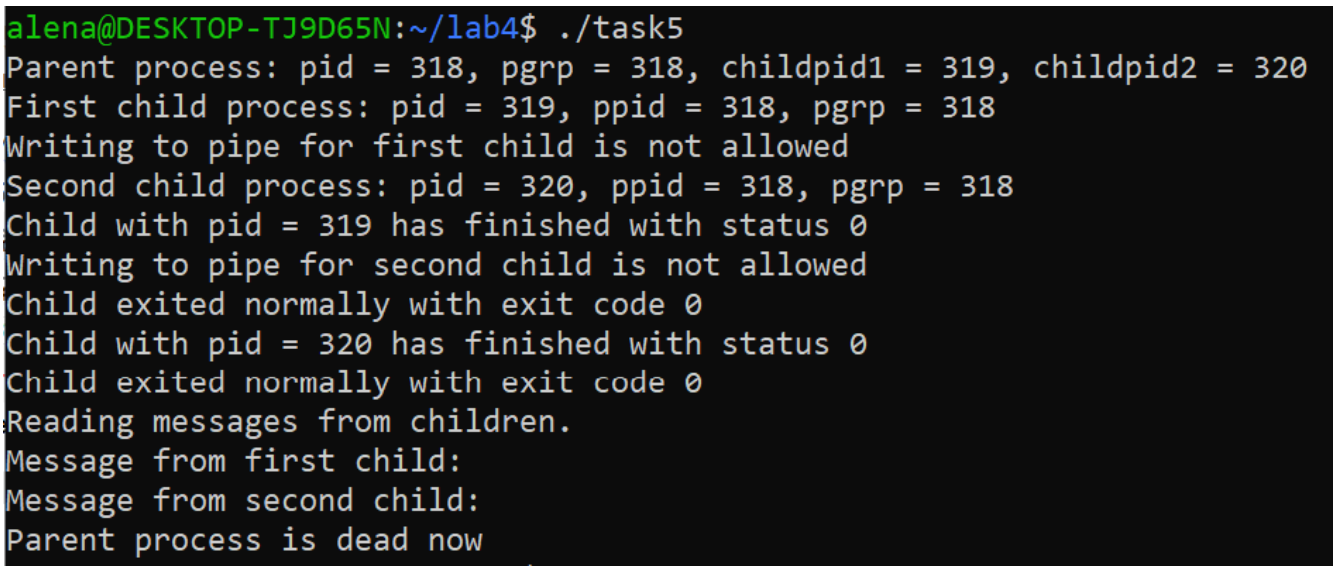

Рис. 6: Демонстрация работы программы при отсутствии отправки сигналов (закомментированы строки 100 и 101).