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

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;
  }
  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.

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
9  #define FORK_OK 0
10 #define FORK_ERR -1
11
```

```
12 #define INTERVAL 1
13
14 int main()
15 {
16   pid_t childpid1, childpid2, childpid;
17   if ((childpid1 = fork()) == FORK_ERR)
18   {
19     perror("Can't fork first child process.\n");
20     return RET_ERR_FORK;
21   }
22   else if (childpid1 == FORK_OK)
23   {
24     printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
25     getpid(), getppid(), getpgrp());
26
27     exit(RET_OK);
28   }
29
30   if ((childpid2 = fork()) == FORK_ERR)
31   {
32     perror("Can't fork second child process.\n");
33     return RET_ERR_FORK;
34   }
35   else if (childpid2 == FORK_OK)
36   {
37     printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
38     getpid(), getppid(), getpgrp());
39
40     exit(RET_OK);
41   }
42
43   sleep(INTERVAL);
44   printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
       d\n",
45   getpid(), getpgrp(), childpid1, childpid2);
46
47   int ch_status;
48   for (int i = 0; i < 2; i++)
49   {
50     childpid = wait(&ch_status);
51     printf("Child with pid = %d has finished with status %d\n", childpid,
         ch_status);
52
53     if (WIFEXITED(ch_status))
54     printf("Child exited normally with exit code %d\n", WEXITSTATUS(
         ch_status));
55     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. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

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 (execlp("echo" , "echo" , "This is echo command from first child",
            NULL) < 0)
28      {
29        perror("Can't execlp 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  }
```

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task3
First child process: pid = 129, ppid = 128, pgrp = 128
Second child process: pid = 130, ppid = 128, pgrp = 128
This is echo command from first child
This is text in file for_cat.txt
Parent process: pid = 128, pgrp = 128, childpid1 = 129, childpid2 = 130
Child with pid = 129 has finished with status 0
Child exited normally with exit code 0
Child with pid = 130 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. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 4: Код программы к заданию №4

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <sys/wait.h>
5  #include <string.h>
6
7  #define RET_OK 0
8  #define RET_ERR_FORK 1
9  #define RET_ERR_PIPE 2
10
11 #define FORK_OK 0
12 #define FORK_ERR -1
13
14 #define INTERVAL 1
15 #define N_CHILDS 2
16 #define MSG1 "This is message from 1 child\n"
17 #define MSG2 "This is message from 2 child\n"
```

```
18 #define LEN12 30
19
20 int main()
21 {
22   pid_t childpid1, childpid2, childpid;
23   int fd[2];
24
25   if (pipe(fd) == -1)
26   {
27     perror("Can't pipe\n");
28     return RET_ERR_PIPE;
29   }
30
31
32   if ((childpid1 = fork()) == FORK_ERR)
33   {
34     perror("Can't fork first child process.\n");
35     return RET_ERR_FORK;
36   }
37   else if (childpid1 == FORK_OK)
38   {
39     printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
40     getpid(), getppid(), getpgrp());
41
42     close(fd[0]);
43     write(fd[1], MSG1, strlen(MSG1) + 1);
44     printf("Message from first child was sent\n");
45
46     exit(RET_OK);
47   }
48
49   if ((childpid2 = fork()) == FORK_ERR)
50   {
51     perror("Can't fork second child process.\n");
52     return RET_ERR_FORK;
53   }
54   else if (childpid2 == FORK_OK)
55   {
56     printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
57     getpid(), getppid(), getpgrp());
58
59     close(fd[0]);
60     write(fd[1], MSG2, strlen(MSG2) + 1);
61     printf("Message from second child was sent\n");
62
63     exit(RET_OK);
64   }
65
66   sleep(INTERVAL);
67   printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
```

```
      d\n",
68    getpid(), getpgrp(), childpid1, childpid2);
69
70    int ch_status;
71    for (int i = 0; i < N_CHILDS; i++)
72    {
73      childpid = wait(&ch_status);
74      printf("Child with pid = %d has finished with status %d\n", childpid,
          ch_status);
75
76      if (WIFEXITED(ch_status))
77      printf("Child exited normally with exit code %d\n", WEXITSTATUS(
          ch_status));
78      else if (WIFSIGNALED(ch_status))
79      printf("Child process ended with a non-intercepted signal number %d\n",
          WTERMSIG(ch_status));
80      else if (WIFSTOPPED(ch_status))
81      printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
          );
82    }
83
84    char message[LEN12] = { 0 };
85
86    printf("Reading messages from children.\n");
87    close(fd[1]);
88
89    for (int i = 0; i < N_CHILDS; i++)
90    {
91
92      if (read(fd[0], message, LEN12) < 0)
93      printf("No messages from child %d.\n", i+1);
94      else
95      printf("Message from child %d:\n%s", i+1, message);
96    }
97
98    printf("Parent process is dead now\n");
99    return RET_OK;
100 }
```

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task4
First child process: pid = 267, ppid = 266, pgrp = 266
Message from first child was sent
Second child process: pid = 268, ppid = 266, pgrp = 266
Message from second child was sent
Parent process: pid = 266, pgrp = 266, childpid1 = 267, childpid2 = 268
Child with pid = 267 has finished with status 0
Child exited normally with exit code 0
Child with pid = 268 has finished with status 0
Child exited normally with exit code 0
Reading messages from children.
Message from child 1:
This is message from 1 child
Message from child 2:
This is message from 2 child
Parent process is dead now
```

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: 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 BIG_INTERVAL 1000
#define SMALL_INTERVAL 800
#define N_CHILDS 2
#define MSG1 "This is message from 1 child\n"
#define MSG2 "This is message from 2 child\n"
#define LEN12 30

```

```
22 void child1_exit(int sig_numb)
23 {
24   printf("First child exits\n");
25   exit(RET_OK);
26 }
27
28 void child2_exit(int sig_numb)
29 {
30   printf("Second child exits\n");
31   exit(RET_OK);
32 }
33
34
35 int main()
36 {
37   pid_t childpid1, childpid2, childpid;
38   int fd[2];
39
40   signal(SIGUSR1, child1_exit);
41   signal(SIGUSR2, child2_exit);
42
43   if (pipe(fd) == -1)
44   {
45     perror("Can't pipe\n");
46     return RET_ERR_PIPE;
47   }
48
49
50   if ((childpid1 = fork()) == FORK_ERR)
51   {
52     perror("Can't fork first child process.\n");
53     return RET_ERR_FORK;
54   }
55   else if (childpid1 == FORK_OK)
56   {
57     printf("First child process: pid = %d, ppid = %d, pgrp = %d\n",
58     getpid(), getppid(), getpgrp());
59
60     close(fd[0]);
61     write(fd[1], MSG1, strlen(MSG1) + 1);
62     printf("Message from first child was sent\n");
63
64     while (1)
65     {
66       printf("First child in infinite loop\n");
67       usleep(SMALL_INTERVAL);
68     }
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      printf("Second child process: pid = %d, ppid = %d, pgrp = %d\n",
79      getpid(), getppid(), getpgrp());
80
81      close(fd[0]);
82      write(fd[1], MSG2, strlen(MSG2) + 1);
83      printf("Message from second child was sent\n");
84
85      while (1)
86      {
87        printf("Second child in infinite loop\n");
88        usleep(SMALL_INTERVAL);
89      }
90    }
91
92    usleep(BIG_INTERVAL);
93    printf("Parent process: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %
         d\n",
94    getpid(), getpgrp(), childpid1, childpid2);
95
96    usleep(BIG_INTERVAL);
97
98    printf("Parent sends signals to stop\n");
99    kill(childpid1, SIGUSR1);
100   kill(childpid2, SIGUSR2);
101
102   int ch_status;
103   for (int i = 0; i < N_CHILDS; i++)
104   {
105     childpid = wait(&ch_status);
106     printf("Child with pid = %d has finished with status %d\n", childpid,
          ch_status);
107
108     if (WIFEXITED(ch_status))
109     printf("Child exited normally with exit code %d\n", WEXITSTATUS(
          ch_status));
110     else if (WIFSIGNALED(ch_status))
111     printf("Child process ended with a non-intercepted signal number %d\n",
          WTERMSIG(ch_status));
112     else if (WIFSTOPPED(ch_status))
113     printf("Child process was stopped by a signal %d\n", WSTOPSIG(ch_status)
          );
114   }
115
116   char message[LEN12] = { 0 };
```

```
117
118   printf("\nReading messages from children.\n");
119   close(fd[1]);
120
121   for (int i = 0; i < N_CHILDS; i++)
122   {
123
124     if (read(fd[0], message, LEN12) < 0)
125     printf("No messages from child %d.\n", i+1);
126     else
127     printf("Message from child %d:\n%s", i+1, message);
128   }
129
130   printf("Parent process is dead now\n");
131   return RET_OK;
132 }
```

```
alena@DESKTOP-TJ9D65N:~/lab4$ ./task5
First child process: pid = 367, ppid = 366, pgrp = 366
Message from first child was sent
First child in infinite loop
Second child process: pid = 368, ppid = 366, pgrp = 366
Parent process: pid = 366, pgrp = 366, childpid1 = 367, childpid2 = 368
Message from second child was sent
First child in infinite loop
Second child in infinite loop
Parent sends signals to stop
First child in infinite loop
Second child exits
First child in infinite loop
First child exits
Child with pid = 368 has finished with status 0
Child exited normally with exit code 0
Child with pid = 367 has finished with status 0
Child exited normally with exit code 0

Reading messages from children.
Message from child 1:
This is message from 1 child
Message from child 2:
This is message from 2 child
Parent process is dead now
```

Рис. 5: Демонстрация работы программы (задание №5).