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: 5 системных вызовов ОС UNIX/LINUX

Дисциплина: Операционные системы

Студент ИУ7-55Б _____ Д.В. Сусликов

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Ю. Рязанова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

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.

Программа представлена в Листинге 1.

Листинг 1 - Программа 1.

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

void print_child_info(int child_num, char *status)
{
  printf("\n%s\n\
  child:\n\
  number = %d\n\
  pid = %d\n\
  parent = %d\n\
  group = %d\n", status, child_num, getpid(),\
   getppid(), getpgrp());
}

pid_t fork_child(int child_num)
{
  pid_t child = fork();
  if (child == -1)
  {
    perror("fork error");
```

```
22          exit(1);
23        }
24        else if (child == 0)
25        {
26          print_child_info(child_num, "S T A R T");
27          sleep(3);
28          print_child_info(child_num, "E N D I N G");
29          exit(0);
30        }
31        return child;
32      }
33
34      int main()
35      {
36        pid_t child_1 = fork_child(1);
37        pid_t child_2 = fork_child(2);
38        pid_t child_3 = fork_child(3);
39
40        printf("\nparent:\n\
41        pid = %d\n\
42        group = %d\n\
43        child_1 = %d\n\
44        child_2 = %d\n\
45        child_3 = %d\n", getpid(), getpgrp(), child_1, child_2,
            child_3);
46
47        return 0;
48      }
```

Результат работы программы показан на Рисунке 1.

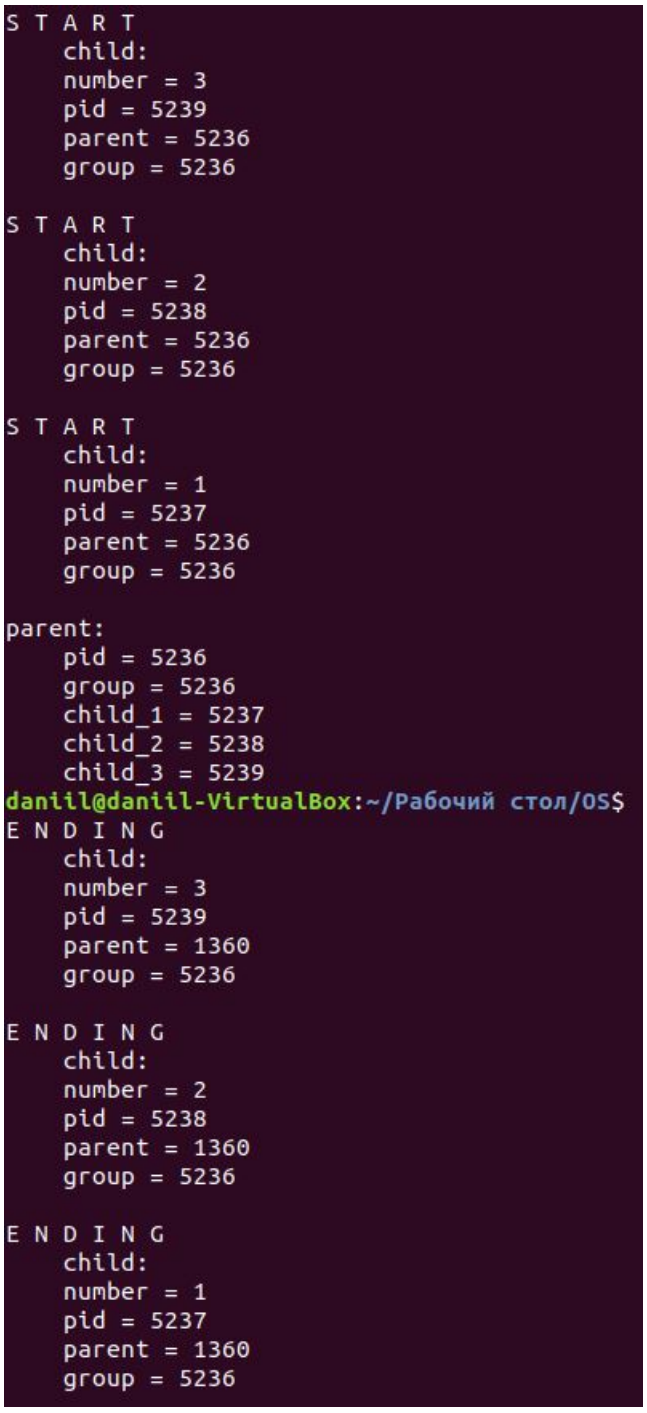

Рисунок 1 – Результат работы программы 1

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.

Программа показана в Листинге 2.

Листинг 2 - Программа 2.

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

void print_child_info(int child_num, char *status)
{
  printf("\n%s\n\
  child:\n\
  number = %d\n\
  pid = %d\n\
  parent = %d\n\
  group = %d\n", status, child_num, getpid(), getppid(), getpgrp
     ());
}

pid_t fork_child(int child_num)
{
  pid_t child = fork();
  if (child == -1)
  {
    perror("fork error");
    exit(1);
  }
  else if (child == 0)
  {
```

```
27          print_child_info(child_num, "S T A R T");
28          sleep(3);
29          print_child_info(child_num, "E N D I N G");
30          exit(0);
31        }
32        return child;
33      }
34
35      void wait_for_child()
36      {
37        int stat_val;
38        pid_t child = wait(&stat_val);
39        printf("Child has finished: PID = %d\n", child);
40        if (WIFEXITED(stat_val))
41          printf("Child = %d ended normal with code %d\n", child,
              WEXITSTATUS(stat_val));
42        else if (WIFSIGNALED(stat_val))
43          printf("Child = %d ended with a non-intercepted signal with
              code %d\n", child, WTERMSIG(stat_val));
44        else if (WIFSTOPPED(stat_val))
45          printf("Child = %d stopped with code %d\n", child, WSTOPSIG(
              stat_val));
46      }
47
48      int main()
49      {
50        pid_t child_1 = fork_child(1);
51        pid_t child_2 = fork_child(2);
52        pid_t child_3 = fork_child(3);
53        printf("\nparent:\n\
54        pid = %d\n\
55        group = %d\n\
56        child_1 = %d\n\
57        child_2 = %d\n\
58        child_3 = %d\n", getpid(), getpgrp(), child_1, child_2,
```

```
            child_3);
59
60          wait_for_child();
61          wait_for_child();
62          wait_for_child();
63
64          return 0;
65      }
```

Результат работы программы изображен на Рисунке 2.

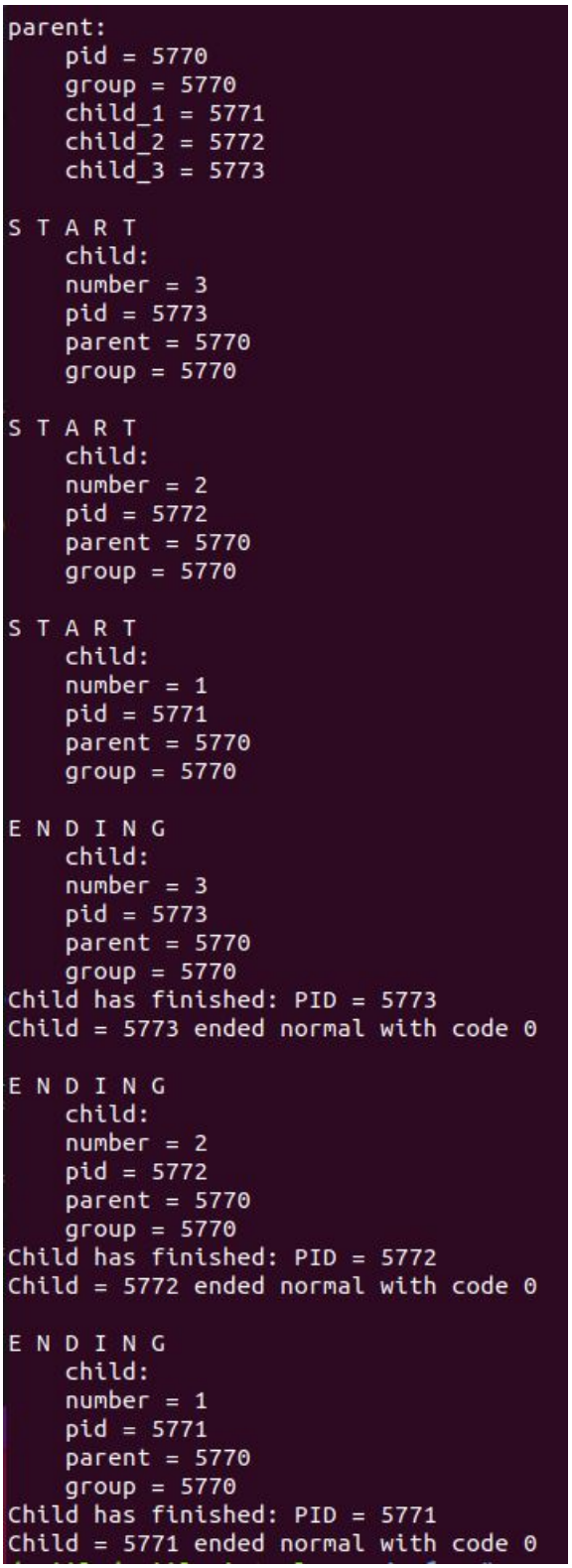

Рисунок 2 – Результат работы программы 2

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.

Программа представлена в Листинге 3.

Листинг 3 - Программа 3.

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

void print_child_info(int child_num)
{
  printf("\nchild:\n\
  number = %d\n\
  pid = %d\n\
  parent = %d\n\
  group = %d\n", child_num, getpid(), getppid(), getpgrp());
}

pid_t fork_child(int child_num, char *path, char *arg0)
{
  pid_t child = fork();
  if (child == -1)
  {
    perror("fork error");
    exit(1);
  }
  else if (child == 0)
  {
    print_child_info(child_num);
    if (execl(path, arg0, NULL) == -1)
```

```
28          {
29            perror("exec error");
30            exit(1);
31          }
32
33        }
34        return child;
35      }
36
37      void wait_for_child()
38      {
39        int stat_val;
40        pid_t child = wait(&stat_val);
41        printf("Child has finished: PID = %d\n", child);
42        if (WIFEXITED(stat_val))
43          printf("Child %d ended normal with code %d\n", child,
              WEXITSTATUS(stat_val));
44        else if (WIFSIGNALED(stat_val))
45          printf("Child %d ended with a non-intercepted signal with
              code %d\n", child, WTERMSIG(stat_val));
46        else if (WIFSTOPPED(stat_val))
47          printf("Child %d stopped with code %d\n", child, WSTOPSIG(
              stat_val));
48      }
49
50      int main()
51      {
52        pid_t child_1 = fork_child(1, "/bin/ls", "ls");
53        pid_t child_2 = fork_child(2, "/bin/ps", "ps");
54
55        printf("\nparent:\n\
56        pid = %d\n\
57        group = %d\n\
58        child_1 = %d\n\
59        child_2 = %d\n", getpid(), getpgrp(), child_1, child_2);
```

```
60
61          wait_for_child();
62          wait_for_child();
63
64          return 0;
65      }
```

Результат работы программы показан на Рисунке 3.

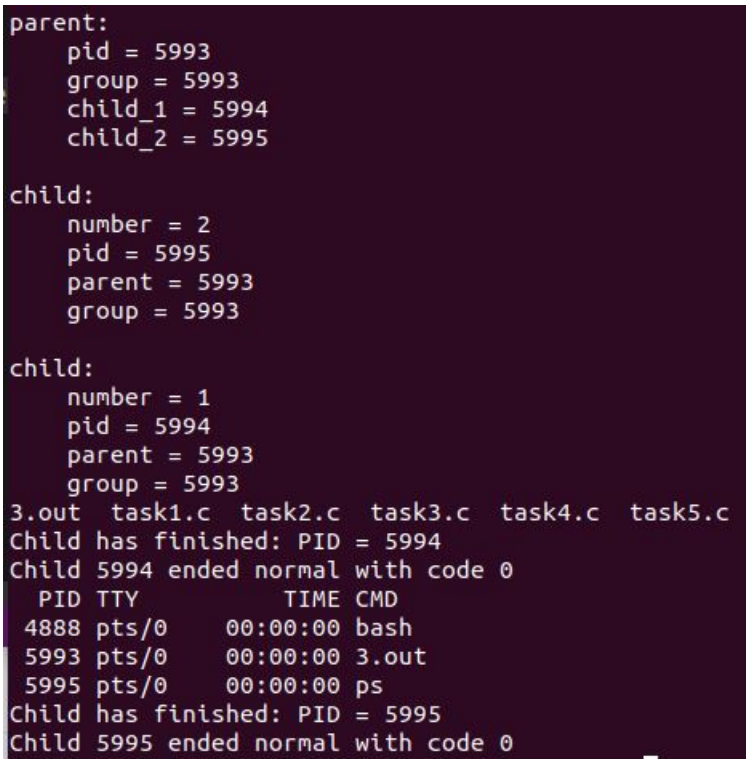

Рисунок 3 – Результат работы программы 3

Задание 4

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

Программа показана в Листинге 4.

Листинг 4 - Программа 4.

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

pid_t fork_child(int child_num, int *fd, char* msg)
{
 pid_t child = fork();
 if (child == -1)
 {
   perror("fork error");
   exit(1);
 }
 else if (child == 0)
 {
   close(fd[0]);
   write(fd[1], msg, sizeof(char) * (strlen(msg) + 1));
   exit(0);
 }
 return child;
}

void wait_for_child(int *fd)
{
 int stat_val;
 char msg[128];
```

```
29
30          close(fd[1]);
31          read(fd[0], msg, sizeof(msg));
32
33          pid_t child = wait(&stat_val);
34          printf("Child %d send %s\n", child, msg);
35          if (WIFEXITED(stat_val))
36          printf("Child %d completed normally with code %d\n", child,
              WEXITSTATUS(stat_val));
37          else if (WIFSIGNALED(stat_val))
38          printf("Child %d ended with a non-intercepted signal with code
               %d\n", child, WTERMSIG(stat_val));
39          else if (WIFSTOPPED(stat_val))
40          printf("Child %d stopped with code %d\n", child, WSTOPSIG(
              stat_val));
41        }
42
43        int main()
44        {
45          int fd[2];
46          if (pipe(fd) == -1)
47          {
48            perror("pipe error");
49            exit(1);
50          }
51
52          pid_t child_1 = fork_child(1, fd, "str 1\0");
53          pid_t child_2 = fork_child(2, fd, "unique string 2\0");
54
55          printf("\nparent:\n\
56          pid = %d\n\
57          group = %d\n\
58          child_1 = %d\n\
59          child_2 = %d\n", getpid(), getpgrp(), child_1, child_2);
60
```

```
61          wait_for_child(fd);
62          wait_for_child(fd);
63
64          return 0;
65      }
```

Результат работы программы изображен на Рисунке 4.

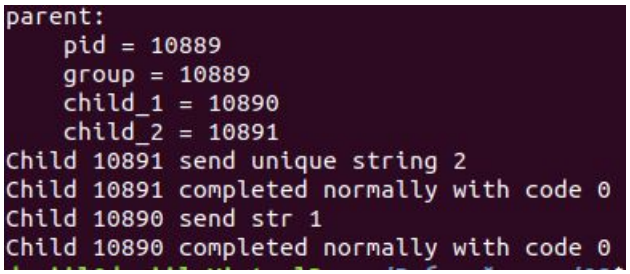

Рисунок 4 – Результат работы программы 4

Задание 5

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

Программа представлена в Листинге 5.

Листинг 5 - Программа 5.

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

int flag = 0;

void catch_signal(int sig_numb)
{
  printf("\nCatch signal Cntrl + Z\n");
  flag = 1;
}

pid_t fork_child(int child_num, int *fd, char* msg)
{
  pid_t child = fork();
  if (child == -1)
  {
    perror("fork error");
    exit(1);
  }
  else if (child == 0)
  {
    if (flag)
    {
      close(fd[0]);
```

```
29            write(fd[1], msg, sizeof(char) * (strlen(msg) + 1));
30          }
31          exit(0);
32        }
33        return child;
34      }
35
36      void wait_for_child(int *fd)
37      {
38        int stat_val;
39        char msg[128];
40
41        pid_t child = wait(&stat_val);
42        if (flag)
43        {
44          close(fd[1]);
45          read(fd[0], msg, sizeof(msg));
46          printf("Child %d send %s\n", child, msg);
47        }
48        else
49        printf("Child %d send NONE\n", child);
50
51        if (WIFEXITED(stat_val))
52        printf("Child %d completed normally with code %d\n", child,
            WEXITSTATUS(stat_val));
53        else if (WIFSIGNALED(stat_val))
54        printf("Child %d ended with a non-intercepted signal with code
            %d\n", child, WTERMSIG(stat_val));
55        else if (WIFSTOPPED(stat_val))
56        printf("Child %d stopped with code %d\n", child, WSTOPSIG(
            stat_val));
57      }
58
59      int main()
60      {
```

```
61      int fd[2];
62
63      if (pipe(fd) == -1)
64      {
65        perror("pipe error");
66        exit(1);
67      }
68
69      printf("Press Ctrl + Z to continue\n");
70      signal(SIGTSTP, catch_signal);
71      sleep(3);
72
73      pid_t child_1 = fork_child(1, fd, "str 1\0");
74      pid_t child_2 = fork_child(2, fd, "unique string 2\0");
75
76      printf("\nparent:\n\
77      pid = %d\n\
78      group = %d\n\
79      child_1 = %d\n\
80      child_2 = %d\n", getpid(), getpgrp(), child_1, child_2);
81
82      wait_for_child(fd);
83      wait_for_child(fd);
84
85      return 0;
86    }
```

Результат работы программы при перехвате сигнала показан на Рисунке 5.

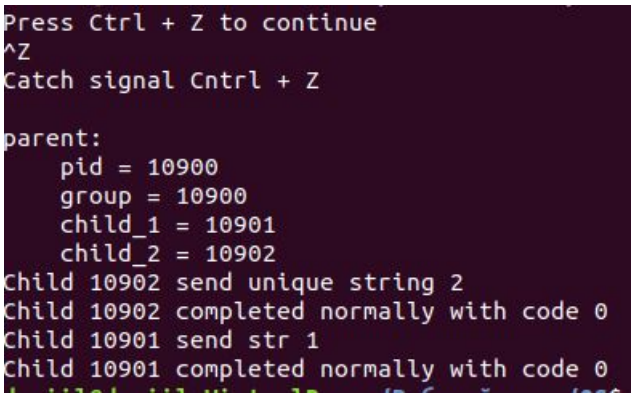

Рисунок 5 – Результат работы программы 5 при перехвате сигнала

Результат работы программы при отсутствии сигнала изображен на Рисунке 6.

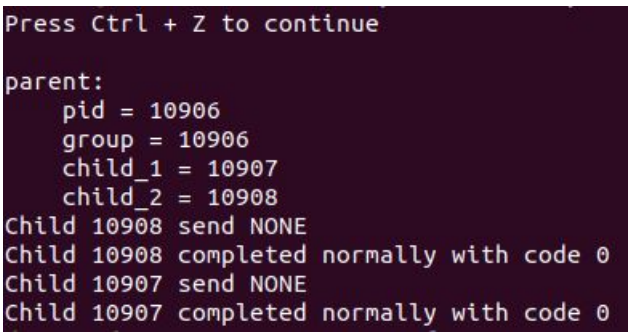

Рисунок 6 – Результат работы программы 6 при отсутствии сигнала