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 Варин Д.В.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва
2021 г.

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 1 – Задание 1

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

int main() {
   const int n = 2;
   int child_processes[n];

   printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n", getpid(), getpgrp());

   for (int i = 0; i < n; i++) {
      child_processes[i] = fork();

      if (child_processes[i] == -1) {
         printf("Can't fork\n");
         return EXIT_FAILURE;
      }

      if (child_processes[i] == 0) {
         sleep(2);
         printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
               getpid(), getppid(), getpgrp());

         return EXIT_SUCCESS;
      }
   }

   printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
      %d\n", \
         getpid(), getpgrp(), child_processes[0], child_processes[1]);

   return EXIT_SUCCESS;
}
```

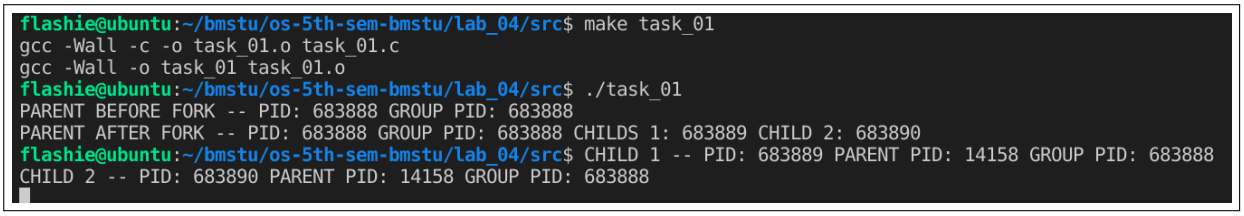

Рисунок 1 – Демонстрация работы программы 1

Можно увидеть, что родительский процесс у потомков имеет PID 14158 - в Ubuntu это процесс посредник(между init с PID = 1 и запущенным в терминале).

Рисунок 2 – Процесс посредник - вывод команды ps -ajx(список системных демонов)

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 – Задание 2

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

int main() {
   const int n = 2;
   int child_processes[n];

   printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n", getpid(), getpgrp());

   for (int i = 0; i < n; i++) {
      child_processes[i] = fork();

      if (child_processes[i] == -1) {
         printf("Can't fork\n");
         return EXIT_FAILURE;
      }

      if (child_processes[i] == 0) {
         sleep(2);
         printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
               getpid(), getppid(), getpgrp());

         return EXIT_SUCCESS;
      }
   }
   for (int i = 0; i < n; i++) {
      int status;

      pid_t child = wait(&status);
      if (WIFEXITED(status)) {
         printf("CHILD PROCESS WITH PID = %d EXITED WITH CODE %d\n", child,
            WEXITSTATUS(status));

      } else {
         printf("CHILD WITH PID = %d PROCESS EXITED UBNORMALLY\n", child);
```

```
38          }
39      }
40      printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
            %d\n", \
41              getpid(), getpgrp(), child_processes[0], child_processes[1]);
42
43      return EXIT_SUCCESS;
44  }
```

```
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_2
PARENT BEFORE FORK -- PID: 790942 GROUP PID: 790942
CHILD 1 -- PID: 790943 PARENT PID: 790942 GROUP PID: 790942
CHILD 2 -- PID: 790944 PARENT PID: 790942 GROUP PID: 790942
CHILD PROCESS WITH PID = 790943 EXITED WITH CODE 0
CHILD PROCESS WITH PID = 790944 EXITED WITH CODE 0
PARENT AFTER FORK -- PID: 790942 GROUP PID: 790942 CHILDS 1: 790943 CHILD 2: 790944
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$
```

Рисунок 3 – Демонстрация работы программы 2

Как можно увидеть, идентификаторы процесса потомка в данном случае на 1 больше процесса предка(или на 1+количество порожденных процессов до текущего), т.к благодаря использованию wait в родительском процессе, усыновления потомков не происходит(как в программе из задания 1).

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.

Задание 4

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

Задание 5

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