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");
         exit(EXIT_FAILURE);
      }

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

   return EXIT_SUCCESS;
}
```

```
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_1
PARENT BEFORE FORK -- PID: 205543 GROUP PID: 205543
PARENT AFTER FORK -- PID: 205543 GROUP PID: 205543 CHILDS 1: 205544 CHILD 2: 205545
PARENT PROCESS FINISHED
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$
CHILD 1 -- PID: 205544 PARENT PID: 4745 GROUP PID: 205543

CHILD 2 -- PID: 205545 PARENT PID: 4745 GROUP PID: 205543
```

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).

```
    1   14158   14158   14158 ?           -1 Ss    1000   0:00 /lib/systemd/systemd --user
```

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");
         exit(EXIT_FAILURE);
      }

      if (child_processes[i] == 0) {
         sleep(1);
         printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
               getpid(), getppid(), getpgrp());
         if (i == n - 1) {
            printf("\n");
         }
         exit(EXIT_SUCCESS);
      }
   }
   printf("\n");
   for (int i = 0; i < n; i++) {
      int status;

      pid_t child = wait(&status);
      printf ("childpid: %d stat_loc: %d \n" , child, status);
      if (WIFEXITED(status)) {
         printf("Child - process finished normally.\n");
```

```
39          printf("CHILD PROCESS WITH PID = %d EXITED WITH CODE %d\n", child,
                WEXITSTATUS(status));
40          printf("\n");
41       } else {
42          printf("Child - process terminates with un - intercepted signal.\n");
43          printf("CHILD WITH PID = %d, TERMINATION CODE = %d\n", child,
                WEXITSTATUS(status));
44          printf("\n");
45       }
46    }
47    printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
          %d\n", \
48          getpid(), getpgrp(), child_processes[0], child_processes[1]);
49    printf("PARENT - PROCESS FINISHED.\n");
50
51    return EXIT_SUCCESS;
52 }
```

```
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_2
PARENT BEFORE FORK -- PID: 234572 GROUP PID: 234572

CHILD 1 -- PID: 234573 PARENT PID: 234572 GROUP PID: 234572
CHILD 2 -- PID: 234574 PARENT PID: 234572 GROUP PID: 234572

childpid: 234573 stat_loc: 0
Child - process finished normally.
CHILD PROCESS WITH PID = 234573 EXITED WITH CODE 0

childpid: 234574 stat_loc: 0
Child - process finished normally.
CHILD PROCESS WITH PID = 234574 EXITED WITH CODE 0

PARENT AFTER FORK -- PID: 234572 GROUP PID: 234572 CHILDS 1: 234573 CHILD 2: 234574
PARENT - PROCESS FINISHED.
```

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.

Листинг 3 – Задание 3

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

int main(int argc, char* argv[]) {
   const int n = 2;
   int child_processes[n];

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

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

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

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

         char arg_child[3];
         sprintf(arg_child, "%d", i + 1);

         int res = execl(argv[1], argv[1], arg_child, 0);
         if (res == -1) {
            printf("Error: can not exec!\n");
            exit(EXIT_FAILURE);
         } else {
            exit(res);
         }

```

```
38          } else {
39              sleep(2);
40          }
41          printf("\n");
42      }
43
44      for (int i = 0; i < n; i++) {
45          int status;
46
47          pid_t child = wait(&status);
48          if (WIFEXITED(status)) {
49              printf("Child - process finished normally.\n");
50              printf("CHILD PROCESS WITH PID = %d EXITED WITH CODE %d\n", child,
                    WEXITSTATUS(status));
51              printf("\n");
52          } else {
53              printf("Child - process terminates with un - intercepted signal.\n");
54              printf("CHILD WITH PID = %d, TERMINATION CODE = %d\n", child,
                    WEXITSTATUS(status));
55              printf("\n");
56          }
57      }
58
59      printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
            %d\n", \
60              getpid(), getpgrp(), child_processes[0], child_processes[1]);
61      printf("PARENT - PROCESS FINISHED.\n");
62
63      return EXIT_SUCCESS;
64  }
```

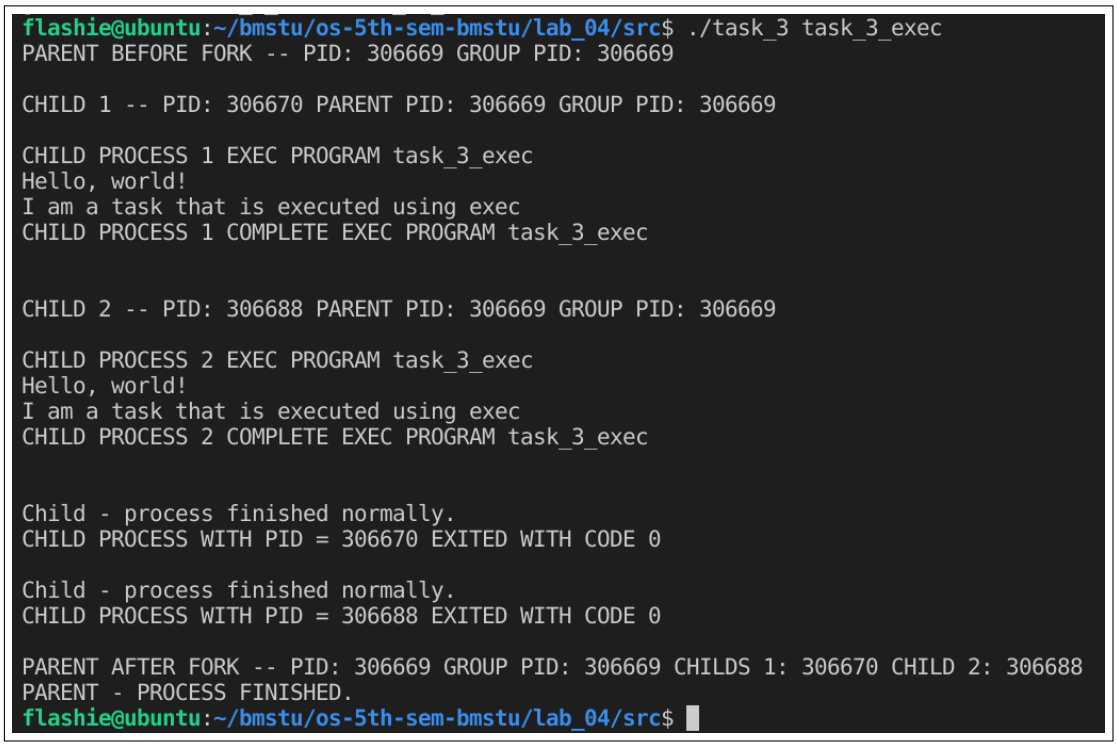

Рисунок 4 – Демонстрация работы программы 3

Задание 4

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

Листинг 4 – Задание 4

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

#define LENBUF 64
const char* message[] = {"dlkbnvlcm", "kfdsnbb ,fkegj"};

int main(int argc, char* argv[]) {
   const int n = 2;
   int child_processes[n];

   char buf[strlen(message[0]) + strlen(message[1]) + 2];

   int fd[2];
   int p = pipe(fd);
   if (p == -1) {
      printf("Can not create pipe\n");
      return EXIT_FAILURE;
   }

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

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

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

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

         close(fd[0]);
```

```
39          write(fd[1], message[i], strlen(message[i]) + 1);
40
41          printf("CHILD %d WITH PID %d send message %s\n\n\n", i+1, getpid(),
                message[i]);
42
43          exit(EXIT_SUCCESS);
44       }
45    }
46
47    for (int i = 0; i < n; i++) {
48       int status;
49
50       pid_t child = wait(&status);
51       if (WIFEXITED(status)) {
52          printf("CHILD PROCESS FINISHED NORMALLY.\n");
53          printf("CHILD PID = %d EXITED WITH CODE %d\n", child,
                WEXITSTATUS(status));
54          printf("\n");
55       } else {
56          printf("CHILD PROCESS TERMINATED WITH UN - INTERCEPTED SIGNAL.\n");
57          printf("CHILD PID = %d, TERMINATION CODE = %d\n", child,
                WEXITSTATUS(status));
58          printf("\n");
59       }
60    }
61
62    printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
          %d\n\n", \
63          getpid(), getpgrp(), child_processes[0], child_processes[1]);
64
65    close(fd[1]);
66    int cur_len = 0;
67    for (int i = 0; i < n; i++) {
68       cur_len += strlen(message[i]) + 1;
69       read(fd[0], buf, cur_len);
70       printf("Message %d - %s\n", i + 1, buf);
71    }
72    printf("\n");
73
74    printf("Child's PID processes which received message with pipe %d %d\n",
75          child_processes[0], child_processes[1]);
76
77    return EXIT_SUCCESS;
78 }
```

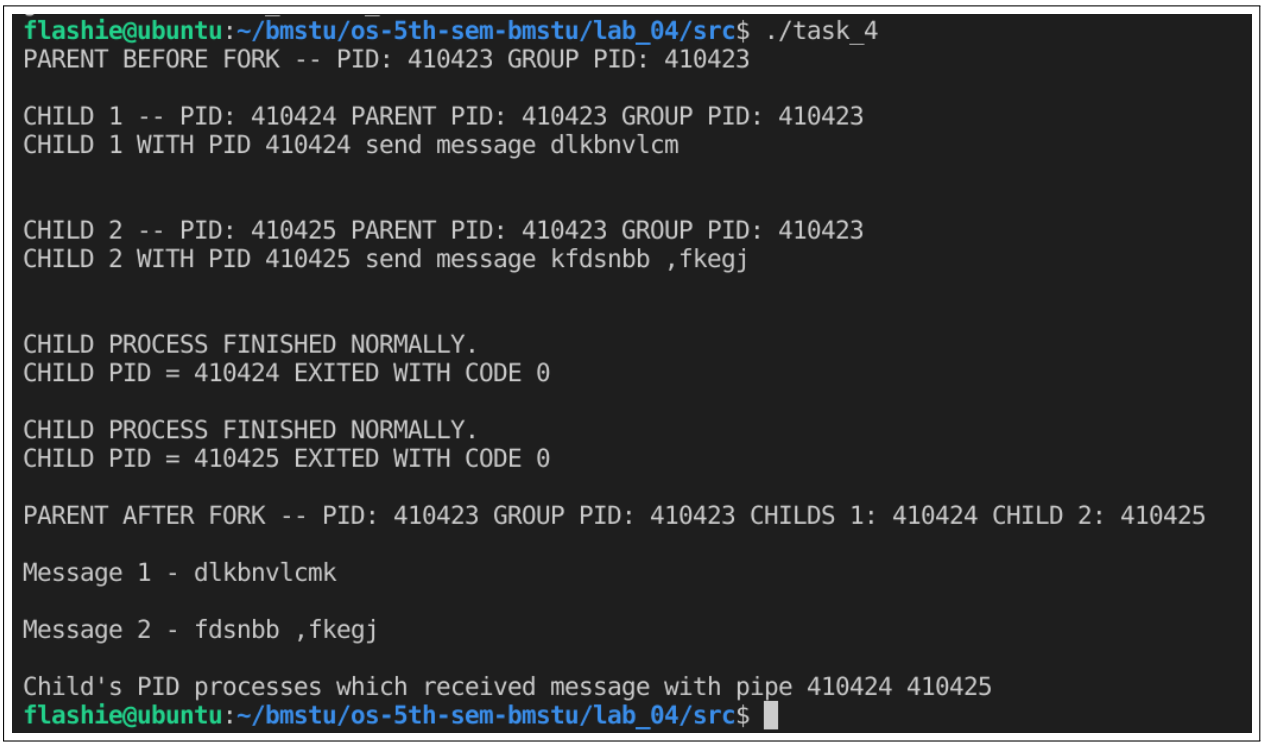

Рисунок 5 – Демонстрация работы программы 4

Задание 5

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

Листинг 5 – Задание 5

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

#define SLEEP 2

#define LENBUF 64
const char* message[] = {"dlkbnvlcm", "kfdsnbb ,fkegj"};
int fd[2];
int i = 0;

int write_to_pipe() {
   close(fd[0]);
   int cur = 0;
   if (getpid() % 2) {
      cur = 1;
   }
   write(fd[1], message[cur], strlen(message[cur]) + 1);
   printf("CHILD PROCESS WITH PID %d send message %s\n", getpid(),
      message[cur]);
   return EXIT_SUCCESS;
}

int main(int argc, char* argv[]) {
   const int n = 2;
   int child_processes[n];

   char buf[LENBUF];

   int p = pipe(fd);
   if (p == -1) {
      printf("Can not create pipe\n");
      return EXIT_FAILURE;
   }

```

```
39    printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n\n", getpid(),
          getpgrp());
40
41    for (int i = 0; i < n; i++) {
42       child_processes[i] = fork();
43
44        if (child_processes[i] == -1) {
45          printf("Can't fork\n");
46          exit(EXIT_FAILURE);
47       }
48
49       if (child_processes[i] == 0) {
50          signal(SIGTSTP, write_to_pipe);
51          sleep(SLEEP);
52          printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
53                getpid(), getppid(), getpgrp());
54          exit(EXIT_SUCCESS);
55       }
56    }
57    signal(SIGTSTP, 1);
58    printf("\n");
59   for (int i = 0; i < n; i++) {
60       int status;
61
62       pid_t child = wait(&status);
63       if (WIFEXITED(status)) {
64          printf("CHILD PROCESS FINISHED NORMALLY.\n");
65          printf("CHILD PID = %d EXITED WITH CODE %d\n", child,
              WEXITSTATUS(status));
66          printf("\n");
67       } else {
68          printf("CHILD PROCESS TERMINATED WITH UN - INTERCEPTED SIGNAL.\n");
69          printf("CHILD PID = %d, TERMINATION CODE = %d\n", child,
              WEXITSTATUS(status));
70          printf("\n");
71       }
72    }
73
74    printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
          %d\n\n", \
75          getpid(), getpgrp(), child_processes[0], child_processes[1]);
76
77    sleep(2);
78
79    close(fd[1]);
```

```
80      for (int i = 0; i < n; i++) {
81          read(fd[0], buf, strlen(message[i]) + 1);
82          printf("Message %d - %s\n", i + 1, buf);
83      }
84      printf("\n");
85
86      printf("PARENT id = %d child_1 pid = %d child_2 pid = %d\n", getpid(),
87             child_processes[0], child_processes[1]);
88
89
90      return EXIT_SUCCESS;
91  }
```

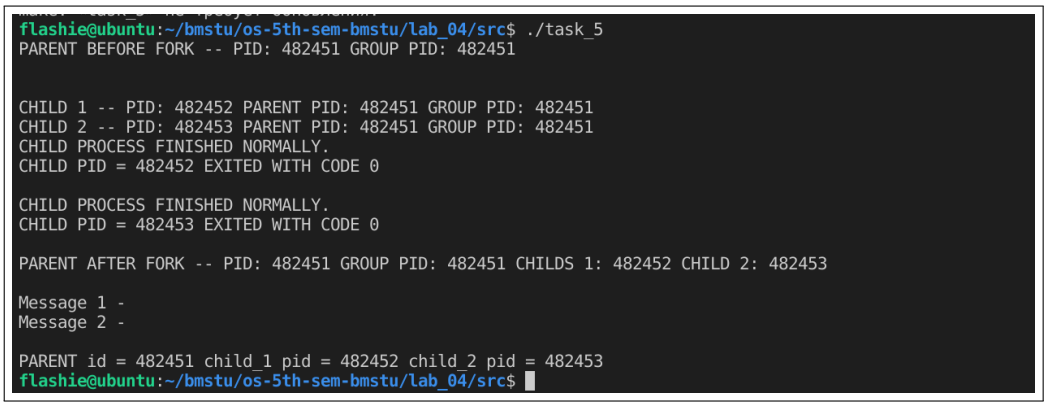

Рисунок 6 – Демонстрация работы программы 5 без посылки сигнала

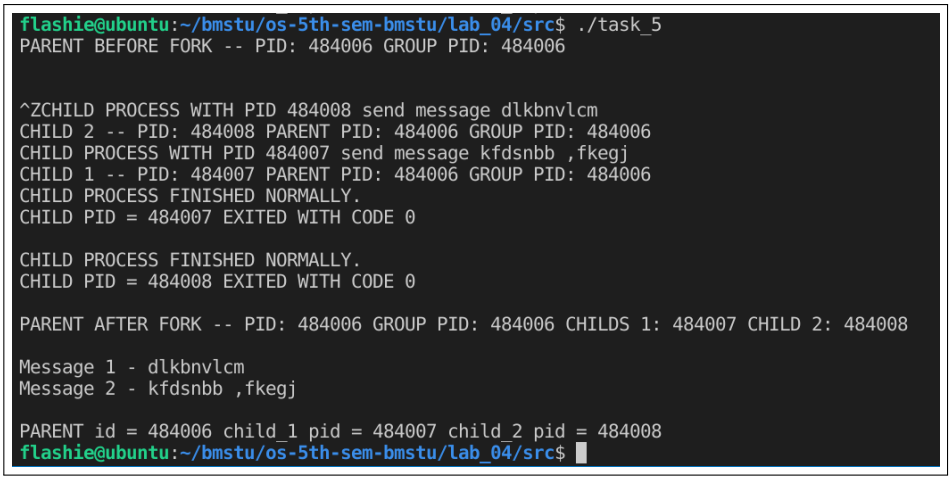

Рисунок 7 – Демонстрация работы программы 5 с сигналом SIGTSTP(сигнал остановки терминала)