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 Шацкий Р.Е.

Группа ИУ7-55Б

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: Процессы-сироты

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

enum error_t {
  no_error,
  fork_failure
};

#define PROC_COUNT 3
#define SLEEP_TIME 1

int main() {
  int children[PROC_COUNT];
  printf("\n Предок —— PID: %d, GROUP: %d\n", getpid(), getpgrp());

  for (int i = 0; i < PROC_COUNT; ++i) {
    int child_pid = fork();

    if (child_pid == -1) {
      perror("Ошибка fork\n");
      return fork_failure;
    }
    else if (child_pid == 0) {
      printf("Для потомка до усыновления    %d —— PID: %d, PPID: %d,
         GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
      sleep(SLEEP_TIME * 4);
      printf("Для потомка после усыновления    %d —— PID: %d, PPID: %
         d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
      return no_error;
    }
    else {
      sleep(SLEEP_TIME);
      children[i] = child_pid;
    }
  }

  printf("Процессы, созданные предком:\n");
  for (int i = 0; i < PROC_COUNT; ++i) {
    printf("Потомок    %d —— PID: %d, ", i + 1, children[i]);
  }
```

```
39          printf("\b\b\nПредок завершился\n");
40
41          return no_error;
42        }
```

```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out

Предок --- PID: 3544, GROUP: 3544
Для потомка до усыновления №1 --- PID: 3545, PPID: 3544, GROUP: 3544
Для потомка до усыновления №2 --- PID: 3546, PPID: 3544, GROUP: 3544
Для потомка до усыновления №3 --- PID: 3547, PPID: 3544, GROUP: 3544
Процессы, созданные предком:
Потомок №1 --- PID: 3545, Потомок №2 --- PID: 3546, Потомок №3 --- PID: 3547,
Предок завершился
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$
Для потомка после усыновления №1 --- PID: 3545, PPID: 1370, GROUP: 3544

Для потомка после усыновления №2 --- PID: 3546, PPID: 1370, GROUP: 3544

Для потомка после усыновления №3 --- PID: 3547, PPID: 1370, GROUP: 3544
```

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: wait()

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include <wait.h>
4
5     enum error_t {
6       no_error,
7       fork_failure
8     };
9
10    #define PROC_COUNT 3
11    #define SLEEP_TIME 1
12
13    int main() {
14      int children[PROC_COUNT];
15      printf("\nПредок —— PID: %d, GROUP: %d\n", getpid(), getpgrp());
16
17      for (int i = 0; i < PROC_COUNT; ++i) {
18        int child_pid = fork();
```

```
19
20          if (child_pid == -1) {
21            perror("Can't fork");
22            return fork_failure;
23          }
24          else if (child_pid == 0) {
25            sleep(SLEEP_TIME);
26            printf("Для потомка    %d —— PID: %d, PPID: %d, GROUP: %d\n", i +
                  1, getpid(), getppid(), getpgrp());
27            return 0;
28          }
29          else {
30            children[i] = child_pid;
31          }
32        }
33
34        printf("Процессы, созданные предком:\n");
35        for (int i = 0; i < PROC_COUNT; ++i) {
36          int status;
37          int stat_value = 0;
38
39          pid_t child_pid = wait(&status);
40          printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
               status);
41
42          if (WIFEXITED(stat_value)) {
43            printf("\tПотомок завершился нормально. Код завершения: %d\n",
                 WEXITSTATUS(stat_value));
44          }
45          else if (WIFSIGNALED(stat_value)) {
46            printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
                 нала: %d\n", WTERMSIG(stat_value));
47          }
48          else if (WIFSTOPPED(stat_value)) {
49            printf("\tПотомок остановился. Номер сигнала: %d\n", WSTOPSIG(
                 stat_value));
50          }
51        }
52        printf("Предок завершился\n");
53
54        return no_error;
55      }
```

```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out

Предок --- PID: 3636, GROUP: 3636
Процессы, созданные предком:
Для потомка №1 --- PID: 3637, PPID: 3636, GROUP: 3636
Потомок с PID = 3637 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0
Для потомка №2 --- PID: 3638, PPID: 3636, GROUP: 3636
Потомок с PID = 3638 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0
Для потомка №3 --- PID: 3639, PPID: 3636, GROUP: 3636
Потомок с PID = 3639 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0
Предок завершился
```

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: exec()

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

enum error_t {
  no_error,
  fork_fail,
  exec_fail
};

#define PROC_COUNT 3
#define SLEEP_TIME 2

int main() {
  int children[PROC_COUNT];
  char *commands[PROC_COUNT] = {"ls", "ps", "pwd"};
  char *args[PROC_COUNT] = {"-al", "ax", "-L"};
  printf("Предок --- PID: %d, GROUP: %d\n", getpid(), getpgrp());

  for (int i = 0; i < PROC_COUNT; ++i) {
```

```
22          int child_pid = fork();
23
24          if (child_pid == -1) {
25            perror("Ошибка fork'a");
26            return fork_fail;
27          }
28          else if (child_pid == 0) {
29            sleep(SLEEP_TIME);
30            printf("Для потомка    %d —— PID: %d, PPID: %d, GROUP: %d\n", i +
                  1, getpid(), getppid(), getpgrp());
31
32            int res = execlp(commands[i], args[i], 0);
33            if (res == -1) {
34              perror("Exec невозможен");
35              return exec_fail;
36            }
37
38            return no_error;
39          }
40          else {
41            children[i] = child_pid;
42          }
43        }
44
45        printf("Процессы, созданные предком:\n");
46        for (int i = 0; i < PROC_COUNT; ++i) {
47          int status;
48          int stat_value = 0;
49
50          pid_t child_pid = wait(&status);
51          printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
               status);
52
53          if (WIFEXITED(stat_value)) {
54            printf("\tПотомок завершился нормально. Код завершения: %d\n",
                 WEXITSTATUS(stat_value));
55          }
56          else if (WIFSIGNALED(stat_value)) {
57            printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
                 нала: %d\n", WTERMSIG(stat_value));
58          }
59          else if (WIFSTOPPED(stat_value)) {
60            printf("\tПотомок остановился. Номер сигнала: %d\n", WSTOPSIG(
                 stat_value));
61          }
62        }
63        printf("Предок завершился\n");
64
65        return no_error;
66      }
```

```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 3691, GROUP: 3691
Процессы, созданные предком:
Для потомка №1 --- PID: 3692, PPID: 3691, GROUP: 3691
Для потомка №2 --- PID: 3693, PPID: 3691, GROUP: 3691
Для потомка №3 --- PID: 3694, PPID: 3691, GROUP: 3691
1_fork.c  2_wait.c  3_exec.c  4_pipe.c  5_signal.c  a.out  wait.o
Потомок с PID = 3692 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0
    PID TTY          TIME CMD
   2503 pts/0    00:00:00 bash
   3691 pts/0    00:00:00 a.out
   3693 pts/0    00:00:00 ps
   3694 pts/0    00:00:00 a.out
Потомок с PID = 3693 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0
/home/wrathen/Documents/bmstu_os_1/lab_4/src
Потомок с PID = 3694 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0
Предок завершился
```

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 4: pipe()

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

enum error_t {
  no_error,
  fork_fail,
  exec_fail,
  pipe_fail
};
```

```
12
13      #define PROC_COUNT 3
14      #define SLEEP_TIME 2
15      #define STR_BUFF_SIZE 64
16
17      int main() {
18        int pipefd[2];  // [0] – чтение, [1] – запись
19        if (pipe(pipefd) == -1) {
20          perror("Ошибка pipe");
21          return pipe_fail;
22        }
23
24        char *msgs[PROC_COUNT] = {"First msg\n", "Second msg\n", "Third
              message\n"};
25        char str_buff[STR_BUFF_SIZE] = {0};
26
27        int children[PROC_COUNT];
28        printf("Предок —— PID: %d, GROUP: %d\n", getpid(), getpgrp());
29
30        for (int i = 0; i < PROC_COUNT; ++i) {
31          int child_pid = fork();
32
33          if (child_pid == -1) {
34            perror("Ошибка fork'a");
35            return fork_fail;
36          }
37          else if (child_pid == 0) {
38            printf("Для потомка    %d —— PID: %d, PPID: %d, GROUP: %d\n", i +
                  1, getpid(), getppid(), getpgrp());
39
40            close(pipefd[0]);
41            write(pipefd[1], msgs[i], strlen(msgs[i]));
42
43            printf("Сообщение    %d отправлено потомку\n\n", i + 1);
44
45            return no_error;
46          }
47          else {
48            children[i] = child_pid;
49          }
50        }
51
52        printf("Процессы, созданные предком:\n");
53        for (int i = 0; i < PROC_COUNT; ++i) {
54          int status;
55          int stat_value = 0;
56
57          pid_t child_pid = wait(&status);
58          printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
              status);
```

```
59
60          if (WIFEXITED(stat_value)) {
61            printf("\tПотомок завершился нормально. Код завершения: %d\n\n",
                 WEXITSTATUS(stat_value));
62          }
63          else if (WIFSIGNALED(stat_value)) {
64            printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
                 нала: %d\n\n", WTERMSIG(stat_value));
65          }
66          else if (WIFSTOPPED(stat_value)) {
67            printf("\tПотомок остановился. Номер сигнала: %d\n\n", WSTOPSIG(
                 stat_value));
68          }
69        }
70
71        close(pipefd[1]);
72        read(pipefd[0], str_buff, STR_BUFF_SIZE);
73        printf("Сообщения, полученные предком: %s", str_buff);
74
75        printf("Предок завершился\n");
76        return no_error;
77      }
```

```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 3741, GROUP: 3741
Процессы, созданные предком:
Для потомка №2 --- PID: 3743, PPID: 3741, GROUP: 3741
Сообщение №2 отправлено потомку

Потомок с PID = 3742 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0

Для потомка №3 --- PID: 3744, PPID: 3741, GROUP: 3741
Сообщение №3 отправлено потомку

Потомок с PID = 3743 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0

Для потомка №1 --- PID: 3742, PPID: 3741, GROUP: 3741
Сообщение №1 отправлено потомку

Потомок с PID = 3744 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0

Сообщения, полученные предком: Second msg
Third message
First msg
Предок завершился
```

Рис. 4: Демонстрация работы программы (задание №4).

Задание 5.

Предок и потомки аналогично №5 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 5: signal()

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

```
6
7     enum error_t {
8       no_error,
9       fork_fail,
10      exec_fail,
11      pipe_fail
12    };
13
14    #define PROC_COUNT 3
15    #define SLEEP_TIME 2
16    #define STR_BUFF_SIZE 64
17
18    void do_nothing(int sigint) {
19      printf("Ничего не происходит?\n");
20    }
21
22    static int sig_status = 0;
23
24    void inc_status(int sigint) {
25      sig_status++;
26      printf("Статус увеличен — %d\n", sig_status);
27    }
28
29    int main() {
30      int pipefd[2];  // [0] — чтение, [1] — запись
31      if (pipe(pipefd) == -1) {
32        perror("Ошибка pipe\n");
33        return pipe_fail;
34      }
35
36      char *msgs[PROC_COUNT] = {"First msg\n", "Second msg\n", "Third
            message\n"};
37      char str_buff[STR_BUFF_SIZE] = {0};
38
39      int children[PROC_COUNT];
40      printf("Предок —— PID: %d, GROUP: %d\n", getpid(), getpgrp());
41      signal(SIGINT, do_nothing);
42
43      for (int i = 0; i < PROC_COUNT; ++i) {
44        int child_pid = fork();
45
46        if (child_pid == -1) {
47          perror("Ошибка fork'a");
48          return fork_fail;
49        }
50        else if (child_pid == 0) {
51          printf("Для потомка    %d —— PID: %d, PPID: %d, GROUP: %d\n", i +
                1, getpid(), getppid(), getpgrp());
52
53          signal(SIGINT, inc_status);
```

```
54            sleep(SLEEP_TIME);
55
56            if (sig_status != 0) {
57              close(pipefd[0]);
58              write(pipefd[1], msgs[i], strlen(msgs[i]));
59              printf("Сообщение    %d отправлено потомку\n\n", i + 1);
60            }
61            else {
62              printf("Сигнала не было\n\n");
63            }
64
65            return no_error;
66          }
67          else {
68            children[i] = child_pid;
69          }
70        }
71
72        printf("Процессы, созданные предком:\n");
73        for (int i = 0; i < PROC_COUNT; ++i) {
74          int status;
75          int stat_value = 0;
76
77          pid_t child_pid = wait(&status);
78          printf("Потомок с PID = %d завершился. Статус: %d\n", children[i],
               status);
79
80          if (WIFEXITED(stat_value)) {
81            printf("\tПотомок завершился нормально. Код завершения: %d\n\n",
                 WEXITSTATUS(stat_value));
82          }
83          else if (WIFSIGNALED(stat_value)) {
84            printf("\tПотомок завершился неперехватываемым сигналом. Номер сиг
                 нала: %d\n\n", WTERMSIG(stat_value));
85          }
86          else if (WIFSTOPPED(stat_value)) {
87            printf("\tПотомок остановился. Номер сигнала: %d\n\n", WSTOPSIG(
                 stat_value));
88          }
89        }
90
91        close(pipefd[1]);
92        read(pipefd[0], str_buff, STR_BUFF_SIZE);
93        printf("Сообщения, полученные предком: %s", str_buff);
94
95        printf("Предок завершился\n");
96        return no_error;
97      }
```

```
wrathen@wrathen-VB:~/Documents/bmstu_os_1/lab_4/src$ ./a.out
Предок --- PID: 3791, GROUP: 3791
Процессы, созданные предком:
Для потомка №2 --- PID: 3793, PPID: 3791, GROUP: 3791
Для потомка №3 --- PID: 3794, PPID: 3791, GROUP: 3791
Для потомка №1 --- PID: 3792, PPID: 3791, GROUP: 3791
Сигнала не было

Потомок с PID = 3792 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0

Сигнала не было

Потомок с PID = 3793 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0

Сигнала не было

Потомок с PID = 3794 завершился. Статус: 0
        Потомок завершился нормально. Код завершения: 0

Сообщения, полученные предком: Предок завершился
```

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