



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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main() {
6     const int n = 2;
7     int child_processes[n];
8
9     printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n", getpid(), getpgrp());
10
11     for (int i = 0; i < n; i++) {
12         child_processes[i] = fork();
13
14         if (child_processes[i] == -1) {
15             printf("Can't fork\n");
16             exit(EXIT_FAILURE);
17         }
18
19         if (child_processes[i] == 0) {
20             sleep(2);
21             printf("\nCHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
22                 getpid(), getppid(), getpgrp());
23             exit(EXIT_SUCCESS);
24         }
25     }
26
27     printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2: \
28         %d\n", \
29         getpid(), getpgrp(), child_processes[0], child_processes[1]);
30     printf("PARENT PROCESS FINISHED\n");
31
32     return EXIT_SUCCESS;
33 }
```

```

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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 int main() {
8     const int n = 2;
9     int child_processes[n];
10
11     printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n", getpid(), getpgrp());
12
13     for (int i = 0; i < n; i++) {
14         child_processes[i] = fork();
15
16         if (child_processes[i] == -1) {
17             printf("Can't fork\n");
18             exit(EXIT_FAILURE);
19         }
20
21         if (child_processes[i] == 0) {
22             sleep(1);
23             printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
24                 getpid(), getppid(), getpgrp());
25             if (i == n - 1) {
26                 printf("\n");
27             }
28             exit(EXIT_SUCCESS);
29         }
30     }
31     printf("\n");
32     for (int i = 0; i < n; i++) {
33         int status;
34
35         pid_t child = wait(&status);
36         printf ("childpid: %d stat_loc: %d \n" , child, status);
37         if (WIFEXITED(status)) {
38             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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <string.h>
7
8 int main(int argc, char* argv[]) {
9     const int n = 2;
10    int child_processes[n];
11
12    printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n\n", getpid(),
13           getpgrp());
14
15    for (int i = 0; i < n; i++) {
16        child_processes[i] = fork();
17
18        if (child_processes[i] == -1) {
19            printf("Can't fork\n");
20            exit(EXIT_FAILURE);
21        }
22
23        if (child_processes[i] == 0) {
24            printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
25                   getpid(), getppid(), getpgrp());
26            printf("\n");
27
28            char arg_child[3];
29            sprintf(arg_child, "%d", i + 1);
30
31            int res = execl(argv[1], argv[1], arg_child, 0);
32            if (res == -1) {
33                printf("Error: can not exec!\n");
34                exit(EXIT_FAILURE);
35            } else {
36                exit(res);
37            }
38        }
39    }
```

```

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,
51             WEXITSTATUS(status));
52         printf("\n");
53     } else {
54         printf("Child - process terminates with un - intercepted signal.\n");
55         printf("CHILD WITH PID = %d, TERMINATION CODE = %d\n", child,
56             WEXITSTATUS(status));
57         printf("\n");
58     }
59
60     printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
61         %d\n", \
62         getpid(), getpgid(), child_processes[0], child_processes[1]);
63     printf("PARENT - PROCESS FINISHED.\n");
64
65     return EXIT_SUCCESS;
66 }

```

```
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_3 task_3_exec
PARENT BEFORE FORK -- PID: 306669 GROUP PID: 306669

CHILD 1 -- PID: 306670 PARENT PID: 306669 GROUP PID: 306669

CHILD PROCESS 1 EXEC PROGRAM task_3_exec
Hello, world!
I am a task that is executed using exec
CHILD PROCESS 1 COMPLETE EXEC PROGRAM task_3_exec

CHILD 2 -- PID: 306688 PARENT PID: 306669 GROUP PID: 306669

CHILD PROCESS 2 EXEC PROGRAM task_3_exec
Hello, world!
I am a task that is executed using exec
CHILD PROCESS 2 COMPLETE EXEC PROGRAM task_3_exec

Child - process finished normally.
CHILD PROCESS WITH PID = 306670 EXITED WITH CODE 0

Child - process finished normally.
CHILD PROCESS WITH PID = 306688 EXITED WITH CODE 0

PARENT AFTER FORK -- PID: 306669 GROUP PID: 306669 CHILDS 1: 306670 CHILD 2: 306688
PARENT - PROCESS FINISHED.
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$
```

Рисунок 4 – Демонстрация работы программы 3

Задание 4

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

Листинг 4 – Задание 4

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <string.h>
7
8 #define LENBUF 64
9 const char* message[] = {"dlkbnvlcm", "kfdsnb ,fkegj"};
10
11 int main(int argc, char* argv[]) {
12     const int n = 2;
13     int child_processes[n];
14
15     char buf[strlen(message[0]) + strlen(message[1]) + 2];
16
17     int fd[2];
18     int p = pipe(fd);
19     if (p == -1) {
20         printf("Can not create pipe\n");
21         return EXIT_FAILURE;
22     }
23
24     printf("PARENT BEFORE FORK -- PID: %d GROUP PID: %d\n\n", getpid(),
25           getppid());
26
27     for (int i = 0; i < n; i++) {
28         child_processes[i] = fork();
29
30         if (child_processes[i] == -1) {
31             printf("Can't fork\n");
32             exit(EXIT_FAILURE);
33         }
34
35         if (child_processes[i] == 0) {
36             printf("CHILD %d -- PID: %d PARENT PID: %d GROUP PID: %d\n", i + 1, \
37                   getpid(), getppid(), getppid());
38
39             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(),
42           message[i]);
43
44     exit(EXIT_SUCCESS);
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,
54               WEXITSTATUS(status));
55         printf("\n");
56     } else {
57         printf("CHILD PROCESS TERMINATED WITH UN - INTERCEPTED SIGNAL.\n");
58         printf("CHILD PID = %d, TERMINATION CODE = %d\n", child,
59               WEXITSTATUS(status));
60         printf("\n");
61     }
62 }
63
64 printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
65       %d\n\n", \
66       getpid(), getpgid(), child_processes[0], child_processes[1]);
67
68 close(fd[1]);
69 int cur_len = 0;
70 for (int i = 0; i < n; i++) {
71     cur_len += strlen(message[i]) + 1;
72     read(fd[0], buf, cur_len);
73     printf("Message %d - %s\n", i + 1, buf);
74 }
75 printf("\n");
76
77 printf("Child's PID processes which received message with pipe %d %d\n",
78       child_processes[0], child_processes[1]);
79
80 return EXIT_SUCCESS;
81 }

```

```
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_4
PARENT BEFORE FORK -- PID: 410423 GROUP PID: 410423

CHILD 1 -- PID: 410424 PARENT PID: 410423 GROUP PID: 410423
CHILD 1 WITH PID 410424 send message dlkbnvlcm

CHILD 2 -- PID: 410425 PARENT PID: 410423 GROUP PID: 410423
CHILD 2 WITH PID 410425 send message kfdsnbb ,fkegj

CHILD PROCESS FINISHED NORMALLY.
CHILD PID = 410424 EXITED WITH CODE 0

CHILD PROCESS FINISHED NORMALLY.
CHILD PID = 410425 EXITED WITH CODE 0

PARENT AFTER FORK -- PID: 410423 GROUP PID: 410423 CHILDS 1: 410424 CHILD 2: 410425

Message 1 - dlkbnvlcmk

Message 2 - fdsnbb ,fkegj

Child's PID processes which received message with pipe 410424 410425
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$
```

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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <string.h>
7 #include <signal.h>
8
9 #define SLEEP 2
10
11 #define LENBUF 64
12 const char* message[] = {"dlkbnvlcm", "kfdsnbb ,fkegj"};
13 int fd[2];
14 int i = 0;
15
16 int write_to_pipe() {
17     close(fd[0]);
18     int cur = 0;
19     if (getpid() % 2) {
20         cur = 1;
21     }
22     write(fd[1], message[cur], strlen(message[cur]) + 1);
23     printf("CHILD PROCESS WITH PID %d send message %s\n", getpid(),
24           message[cur]);
25     return EXIT_SUCCESS;
26 }
27
28 int main(int argc, char* argv[]) {
29     const int n = 2;
30     int child_processes[n];
31
32     char buf[LENBUF];
33
34     int p = pipe(fd);
35     if (p == -1) {
36         printf("Can not create pipe\n");
37         return EXIT_FAILURE;
38     }
```

```

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,
66              WEXITSTATUS(status));
67          printf("\n");
68      } else {
69          printf("CHILD PROCESS TERMINATED WITH UN - INTERCEPTED SIGNAL.\n");
70          printf("CHILD PID = %d, TERMINATION CODE = %d\n", child,
71              WEXITSTATUS(status));
72          printf("\n");
73      }
74  }
75
76  printf("PARENT AFTER FORK -- PID: %d GROUP PID: %d CHILDS 1: %d CHILD 2:
    %d\n\n", \
77      getpid(), getpgrp(), child_processes[0], child_processes[1]);
78
79  sleep(2);
80
81  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 }

```

```

flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_5
PARENT BEFORE FORK -- PID: 482451 GROUP PID: 482451

CHILD 1 -- PID: 482452 PARENT PID: 482451 GROUP PID: 482451
CHILD 2 -- PID: 482453 PARENT PID: 482451 GROUP PID: 482451
CHILD PROCESS FINISHED NORMALLY.
CHILD PID = 482452 EXITED WITH CODE 0

CHILD PROCESS FINISHED NORMALLY.
CHILD PID = 482453 EXITED WITH CODE 0

PARENT AFTER FORK -- PID: 482451 GROUP PID: 482451 CHILDS 1: 482452 CHILD 2: 482453

Message 1 -
Message 2 -

PARENT id = 482451 child_1 pid = 482452 child_2 pid = 482453
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$

```

Рисунок 6 – Демонстрация работы программы 5 без отправки сигнала

```

flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$ ./task_5
PARENT BEFORE FORK -- PID: 484006 GROUP PID: 484006

^ZCHILD PROCESS WITH PID 484008 send message dlkbnvlcm
CHILD 2 -- PID: 484008 PARENT PID: 484006 GROUP PID: 484006
CHILD PROCESS WITH PID 484007 send message kfdsnb ,fkegj
CHILD 1 -- PID: 484007 PARENT PID: 484006 GROUP PID: 484006
CHILD PROCESS FINISHED NORMALLY.
CHILD PID = 484007 EXITED WITH CODE 0

CHILD PROCESS FINISHED NORMALLY.
CHILD PID = 484008 EXITED WITH CODE 0

PARENT AFTER FORK -- PID: 484006 GROUP PID: 484006 CHILDS 1: 484007 CHILD 2: 484008

Message 1 - dlkbnvlcm
Message 2 - kfdsnb ,fkegj

PARENT id = 484006 child_1 pid = 484007 child_2 pid = 484008
flashie@ubuntu:~/bmstu/os-5th-sem-bmstu/lab_04/src$

```

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