



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»*

**О Т Ч Е Т**

по лабораторной работе № 0 4

**Название** *Процессы. Системные вызовы fork() и exec().*

**Дисциплина:** *Операционные системы*

Студент

*ИУ7И-52Б*

(Группа)

**Чыонг Н. В. У.**

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

**Рязанова Н. Ю.**

(Подпись, дата)

(И.О. Фамилия)

Москва, 2021г

### ❖ 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.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <unistd.h>
5.
6. #define DELAY 4
7.
8. int main() {
9.     pid_t child1 = fork();
10.
11.     if (child1 == -1)
12.     {
13.         perror("Can't fork Child 1");
14.         exit(1);
15.     }
16.     else if (child1 == 0) // Child 1
17.     {
18.         printf("Child 1: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
19.         sleep(DELAY);
20.         printf("Child 1: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
21.         return 0;
22.     }
23.
24.     pid_t child2 = fork();
25.     if (child2 == -1)
26.     {
27.         perror("Can't fork");
28.         exit(1);
29.     }
30.     else if (child2 == 0) // Child 2
31.     {
32.         printf("Child 2: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
33.         sleep(DELAY);
34.         printf("Child 2: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
35.         return 0;
36.     }
37.
38.     // Parent
39.     printf("Parent: pid=%d, child1=%d, child2=%d, groupid=%d\n", getpid(), child1, child2,
40.         getpgrp());
41.     return 0;
42. }
```

```
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04$ ./task_1.exe
Parent: pid=8175, child1=8176, child2=8177, groupid=8175
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04$ Child 1: pid=8176, ppid=1111, groupid=8175
Child 2: pid=8177, ppid=1111, groupid=8175
Child 1: pid=8176, ppid=1111, groupid=8175
Child 2: pid=8177, ppid=1111, groupid=8175
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04$ █
```

## ❖ 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.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <sys/wait.h>
5. #include <unistd.h>
6.
7. #define DELAY 5
8.
9. int main() {
10.     // Fork Child 1
11.     pid_t child1 = fork();
12.     if (child1 == -1)
13.     {
14.         perror("Couldn't fork.");
15.         exit(1);
16.     }
17.     else if (child1 == 0)
18.     {
19.         printf("Child 1: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp() );
20.         sleep(DELAY);
21.         printf("Child 1: Exiting\n");
22.         return 0;
23.     }
24.
25.     // Fork Child 2
26.     pid_t child2 = fork();
27.     if (child2 == -1)
28.     {
29.         perror("Couldn't fork.");
30.         exit(1);
31.     }
32.     else if (child2 == 0)
33.     {
34.         printf("Child 2: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
35.         sleep(DELAY);
36.         printf("Child 2: Exiting\n");
37.         return 0;
38.     }
39.
40.     // Parent
41.     int status1;
42.     pid_t ret1 = wait(&status1);
43.     int status2;
44.     pid_t ret2 = wait(&status2);
45.
46.     printf("Parent: pid = %d, group = %d, child1 = %d, Child2 = %d\n", getpid(), getpgrp(), child1, child2 );
47.
48.     if (WIFEXITED(status1))
49.         printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
50.     else if (WIFSIGNALED(status1))
51.         printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
52.     else if (WIFSTOPPED(status1))
53.         printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
54.
55.     if (WIFEXITED(status2))
56.         printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
57.     else if (WIFSIGNALED(status2))
58.         printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
59.     else if (WIFSTOPPED(status2))
60.         printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
61.
62.     return 0;
63. }
```

```

tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_2$ gcc -std=c99 -o task_2.exe task_2.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_2$ ./task_2.exe
Child 1: pid = 9032, ppid = 9031, groupid = 9031
Child 2: pid = 9033, ppid = 9031, groupid = 9031
Child 1: Exiting
Child 2: Exiting
Parent: pid = 9031, group = 9031, Child1 = 9032, Child2 = 9033
Parent: child 9032 finished with 0 code.
Parent: child 9033 finished with 0 code.
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_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.

```

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.     // Child 1
9.     pid_t child1 = fork();
10.    if (child1 == -1)
11.    {
12.        perror("Child 1: Couldn't fork.");
13.        exit(1);
14.    }
15.    else if (child1 == 0)
16.    {
17.        printf("Child 1: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
18.
19.        if (execlp("ps", "ps", "al", 0) == -1)
20.        {
21.            perror("Child 1: couldn't exec.");
22.            exit(1);
23.        }
24.    }
25.    // Child 2
26.    pid_t child2 = fork();
27.    if (child2 == -1)
28.    {
29.        perror("Child 2: Couldn't fork.");
30.        exit(1);
31.    }
32.    else if (child2 == 0)
33.    {
34.        printf("Child 2: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
35.
36.        if (execlp("ls", "ls", "-ail", 0) == -1)
37.        {
38.            perror("Child 2: couldn't exec.");
39.            exit(1);
40.        }
41.    }
42.    // Parent
43.    if (child1 != 0 && child2 != 0)
44.    {
45.        int status1, status2;
46.        pid_t ret1 = wait(&status1);
47.        pid_t ret2 = wait(&status2);
48.
49.        printf("Parent: pid=%d, groupid=%d, child1=%d, child2=%d\n", getpid(), getpgrp(), child1, child
50.        2);
51.
52.        if (WIFEXITED(status1))
53.            printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));

```

```

53.         else if (WIFSIGNALED(status1))
54.             printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
55.         else if (WIFSTOPPED(status1))
56.             printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
57.
58.         if (WIFEXITED(status2))
59.             printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
60.         else if (WIFSIGNALED(status2))
61.             printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
62.         else if (WIFSTOPPED(status2))
63.             printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
64.     }
65.
66.     return 0;
67. }

```

```

tnvu4920@tnvu4920-VirtualBox:~/lu7-52b/lab_04/task_3$ vim task_3.c
tnvu4920@tnvu4920-VirtualBox:~/lu7-52b/lab_04/task_3$ gcc -std=c99 -o task_3.exe task_3.c
tnvu4920@tnvu4920-VirtualBox:~/lu7-52b/lab_04/task_3$ ./task_3.exe
Child 1: pid = 9103, ppid = 9102, groupid = 9102
Child 2: pid = 9104, ppid = 9102, groupid = 9102
total 32
934518 drwxrwxr-x 2 tnvu4920 tnvu4920 4096 ноя 19 03:24 .
798912 drwxrwxr-x 5 tnvu4920 tnvu4920 4096 ноя 19 03:15 ..
934521 -rw-rw-r-- 1 tnvu4920 tnvu4920 1759 ноя 19 03:23 task_3.c
934519 -rw-rwxr-x 1 tnvu4920 tnvu4920 17096 ноя 19 03:24 task_3.exe
F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND
4 1000 1192 1103 20 0 172652 6460 poll_s Ssl+ tty2 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
4 1000 1195 1192 20 0 627748 140808 ep_poll Sll+ tty2 0:59 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -nolisten tcp -verbose 3
0 1000 1254 1192 20 0 199304 15060 poll_s Sll+ tty2 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
0 1000 1888 1849 20 0 19508 4980 do_wai Ss pts/0 0:00 bash
0 1000 7844 1888 20 0 2496 584 do_sig T pts/0 0:04 ./app.exe
1 1000 7845 7844 20 0 2496 76 do_sig T pts/0 0:04 ./app.exe
0 1000 7911 1888 20 0 2496 580 do_sig T pts/0 0:02 ./app.exe
1 1000 7912 7911 20 0 2496 80 do_sig T pts/0 0:02 ./app.exe
0 1000 9102 1888 20 0 2364 588 do_wai S+ pts/0 0:00 ./task_3.exe
4 1000 9103 9102 20 0 20852 3256 - R+ pts/0 0:00 ps al
Parent: pid=9102, groupid=9102, child1=9103, child2=9104
Parent: child 9104 finished with 0 code.
Parent: child 9103 finished with 0 code.
tnvu4920@tnvu4920-VirtualBox:~/lu7-52b/lab_04/task_3$

```

#### ❖ Задание 4:

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

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <string.h>
5. #include <sys/types.h>
6. #include <sys/wait.h>
7.
8. #define LEN 100
9.
10. int main() {
11.     // Pipe
12.     int fd[2];
13.     if (pipe(fd) == -1)
14.     {
15.         perror("Couldn't pipe.");
16.         exit(1);
17.     }
18.     // Child 1
19.     pid_t child1 = fork();
20.     if (child1 == -1)
21.     {
22.         perror("Couldn't fork.");
23.         exit(1);
24.     }
25.     else if (child1 == 0)
26.     {
27.         close(fd[0]);
28.         char msg1[] = "Hello From Child 1 ";
29.         write(fd[1], msg1, LEN);
30.         printf("Child 1: writen [%s] to Pipe\n", msg1);
31.         exit(0);
32.     }
33.     // Child 2
34.     int child2 = fork();
35.     if (child2 == -1)
36.     {
37.         perror("Couldn't fork.");
38.         exit(1);
39.     }
40.     else if (child2 == 0)
41.     {
42.         close(fd[0]);
43.         char msg2[] = "Hello From Child 2 ";
44.         write(fd[1], msg2, LEN);
45.         printf("Child 2: writen [%s] to Pipe\n", msg2);
46.         exit(0);
47.     }
48.     // Parentss
49.
50.     int status1, status2;
51.     pid_t ret1 = wait(&status1);
52.     pid_t ret2 = wait(&status2);
53.
54.     close(fd[1]);
55.     char msg1[LEN];
56.     read(fd[0], msg1, LEN);
57.     char msg2[LEN];
58.     read(fd[0], msg2, LEN);
59.     printf("Parent: read from Pipe [%s%s]\n", msg1, msg2);
60.     if (WIFEXITED(status1))
61.         printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
62.     else if (WIFSIGNALED(status1))
63.         printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
64.     else if (WIFSTOPPED(status1))
65.         printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
66. }
```

```

67.     if (WIFEXITED(status2))
68.         printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
69.     else if (WIFSIGNALED(status2))
70.         printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
71.     else if (WIFSTOPPED(status2))
72.         printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
73.     return 0;
74. }

```

```

tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_4$ vim task_4.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_4$ gcc -std=c99 -o task_4.exe task_4.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_4$ ./task_4.exe
Child 1: writen [Hello From Child 1 ] to Pipe
Child 2: writen [Hello From Child 2 ] to Pipe
Parent: read from Pipe [Hello From Child 1 Hello From Child 2 ]
Parent: child 9189 finished with 0 code.
Parent: child 9190 finished with 0 code.
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_4$

```

### ❖ Задание 5:

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

```

1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <string.h>
5.  #include <signal.h>
6.  #include <time.h>
7.  #include <sys/wait.h>
8.
9.  #define LEN 100
10. #define DELAY 2
11.
12. int p_flag = 0;
13.
14. void catch_sig(int signum) {
15.     printf("Proccess %d: catched signal %d\n", getpid(), signum);
16.     p_flag = 1;
17. }
18.
19. int main() {
20.     //Pipe
21.     int fd[2];
22.     if (pipe(fd) == -1)
23.     {
24.         perror("Couldn't pipe.");
25.         exit(1);
26.     }
27.     // Ctrl + Z
28.     void (*old_handler)(int) = signal(SIGTSTP, catch_sig);
29.
30.     // Child 1
31.     pid_t child1 = fork();
32.     if (child1 == -1)
33.     {
34.         perror("Couldn't fork.");
35.         exit(1);
36.     }
37.     else if (child1 == 0)
38.     {
39.         while (!p_flag) ;
40.         close(fd[0]);
41.         char msg1[] = "Hello From Child 1 ";
42.         write(fd[1], msg1, LEN);
43.         printf("Child 1: writen [%s] to Pipe\n", msg1);
44.     }
45.
46.     // Child 2
47.     pid_t child2 = fork();

```

```

48.     if (child2 == -1)
49.     {
50.         perror("Couldn't fork.");
51.         exit(1);
52.     }
53.     else if (child2 == 0)
54.     {
55.         while (!p_flag) ;
56.         close(fd[0]);
57.         char msg2[] = "Hello From Child 2 ";
58.         write(fd[1], msg2, LEN);
59.         printf("Child 2: writen [%s] to Pipe\n", msg2);
60.     }
61.     // Parent
62.     if (child1 != 0 && child2 != 0)
63.     {
64.         printf("Parent: pid = %d\n", getpid());
65.         printf("Child 1: pid = %d\n", child1);
66.         printf("Child 2: pid = %d\n\n", child2);
67.         printf("Parent: waiting for CTRL+Z signal\n");
68.         if  = (!p_flag);
69.
70.         int status1, status2;
71.         pid_t ret1 = wait(&status1);
72.         pid_t ret2 = wait(&status2);
73.
74.         printf("-----\n");
75.         close(fd[1]);
76.         char msg1[LEN], msg2[LEN];
77.         read(fd[0], msg1, LEN);
78.         read(fd[0], msg2, LEN);
79.         printf("Parent: read from Pipe [%s%s]\n", msg1, msg2);
80.
81.         if (WIFEXITED(status1))
82.             printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
83.         else if (WIFSIGNALED(status1))
84.             printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
85.         else if (WIFSTOPPED(status1))
86.             printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
87.
88.         if (WIFEXITED(status2))
89.             printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
90.         else if (WIFSIGNALED(status2))
91.             printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
92.         else if (WIFSTOPPED(status2))
93.             printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
94.     }
95.     signal(SIGTSTP, old_handler);
96.     return 0;
97. }

```

```

tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_5$ vim task_5.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_5$ gcc -std=c99 -o task_5.exe task_5.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_5$ ./task_5.exe
Parent: pid = 9372
Child 1: pid = 9373
Child 2: pid = 9374

Parent: waiting for CTRL+Z signal
^ZProcess 9372: caught signal 20
Process 9374: caught signal 20
Child 2: writen [Hello From Child 2 ] to Pipe
Process 9373: caught signal 20
Child 1: writen [Hello From Child 1 ] to Pipe
-----
Parent: read from Pipe [Hello From Child 2 Hello From Child 1 ]
Parent: child 9374 finished with 0 code.
Parent: child 9373 finished with 0 code.
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_5$

```