

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №5

По курсу: "ОПЕРАЦИОННЫЕ СИСТЕМЫ"

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

Работу выполнил: Мокеев Даниил, ИУ7-56

Преподаватели: Рязанова Н.Ю.

Москва, 2019

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Постановка задачи	3
Введение	3
2 Технологическая часть	4
2.1 Листинг кода алгоритмов	4
2.2 Примеры работы программ	12
Вывод	15
Заключение	16

Введение

Целью данной лабораторной работы является изучение системных вызовов `fork()` и `exec()`, программных каналов и сигналов, получение навыков их использования.

Задачи данной лабораторной работы:

- изучить системные вызовы `exec()` и `fork()`;
- рассмотреть программные каналы и сигналы;
- реализовать пять программ по заданию.

1 | Аналитическая часть

В данной части будут рассмотрена постановка задачи практической части лабораторной работы.

1.1 Постановка задачи

В ходе лабораторной необходимо реализовать пять программ со следующими требованиями:

- 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.
- предок ждет завершения своих потомков, используя системный вызов `wait()`. Вывод соответствующих сообщений на экран.
- потомки переходят на выполнение других программ. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.
- предок и потомки обмениваются сообщениями через неименованный программный канал. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.
- предок и потомки обмениваются сообщениями через неименованный программный канал. С помощью сигнала меняется ход выполнения программы. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран

Вывод

В данном разделе были рассмотрены общие требования к программам, реализуемым в ходе выполнения лабораторной работы.

2 | Технологическая часть

2.1 Листинг кода алгоритмов

В данном разделе будут приведены листинги кода реализованных программ.

Листинг 2.1: Процессы-сироты

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     int child_1, child_2;
6     child_1 = fork();
7     if(child_1 == -1){
8         perror("Coulnd't fork child #1");
9         exit(1);
10    }
11    if (child_1 == 0){
12        sleep(1);
13        printf("Child #1: pid=%d; group=%d; ppid=%d\n",
14            getpid(), getpgrp(), getppid());
15
16        return 0;
17    }
18    if (child_1 > 0){
19        child_2 = fork();
20        if(child_2 == -1){
21            perror("Coulnd't fork child #2");
22            exit(1);
23        }
24        if (child_2 == 0){
25            printf("\nChild #2: pid=%d; group=%d; ppid=%d\n",
26                getpid(), getpgrp(), getppid());
27
28            return 0;
29        }else{
30            printf("Parent: pid=%d; group=%d; ppid=%d\n",
31                getpid(), getpgrp(), getppid());
32
33            return 0;
34        }
35    }
36 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5
6 int main(){
7     int child_1, child_2;
8     child_1 = fork();
9     if(child_1 == -1){
10         perror("Coulnd't fork child #1");
11         exit(1);
12     }
13     if (child_1 == 0){
14         sleep(1);
15         printf("Child #1: pid=%d; group=%d; ppid=%d\n",
16             getpid(), getpgrp(), getppid());
17
18         return 0;
19     }
20     if (child_1 > 0){
21         child_2 = fork();
22         if(child_2 == -1){
23             perror("Coulnd't fork child #2");
24             exit(1);
25         }
26         if (child_2 == 0){
27             printf("\nChild #2: pid=%d; group=%d; ppid=%d\n",
28                 getpid(), getpgrp(), getppid());
29
30             return 0;
31         }else{
32             sleep(2);
33             printf("Parent: pid=%d; group=%d; ppid=%d\n",
34                 getpid(), getpgrp(), getppid());
35
36             pid_t child_pid;
37             int status;
38
39             child_pid = wait(&status);
40             if (WIFEXITED(status))
41                 printf("Parent: child %d finished with code %d\n",
42                     child_pid, WEXITSTATUS(status) );
43             else if (WIFSTOPPED(status))
44                 printf("Parent: child %d finished with code %d\n",
45                     child_pid, WSTOPSIG(status) );
46             return 0;
47         }
48     }
```

49

50 }

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     int child_1, child_2;
7     child_1 = fork();
8     if(child_1 == -1){
9         perror("Coulnd't fork child #1");
10        exit(1);
11    }
12    if (child_1 == 0){
13        sleep(1);
14        printf("\nChild #1 executes ps -al\n\n");
15        if(execlp("ps", "ps", "-al", (char*)NULL) == -1){
16            printf("Couldn't exec ps command\n");
17            exit(1);
18        }
19
20        return 0;
21    }
22    if (child_1 > 0){
23        child_2 = fork();
24        if(child_2 == -1){
25            perror("Coulnd't fork child #2");
26            exit(1);
27        }
28        if (child_2 == 0){
29            printf("\nChild #2 executes ls -l\n\n");
30            if(execlp("ls", "ls", "-l", (char*)NULL) == -1){
31                printf("Couldn't exec ps command\n");
32                exit(1);
33            }
34            return 0;
35        }else{
36            printf("\nParent: pid=%d; group=%d; ppid=%d\n",
37                getpid(), getpgrp(), getppid());
38
39            pid_t child_pid;
40            int status;
41
42            //waiting for the second child to finish
43            child_pid = wait(&status);
44            if (WIFEXITED(status))
45                printf("\nParent: child with pid = %d finished with code %d\n",
46                    child_pid, WEXITSTATUS(status) );
47            else if (WIFSTOPPED(status))
48                printf("\nParent: child %d finished with code %d\n",
```



```
49         child_pid, WSTOPSIG(status) );
50
51     //waiting for the first child to finish
52     child_pid = wait(&status);
53     if (WIFEXITED(status))
54         printf("\nParent: child with pid = %d finished with code %d\n",
55             child_pid, WEXITSTATUS(status) );
56     else if (WIFSTOPPED(status))
57         printf("\nParent: child %d finished with code %d\n",
58             child_pid, WSTOPSIG(status) );
59
60     return 0;
61     }
62 }
63 }
```

```
1  /*
2  exchanging messages with parent
3  */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #define MESSAGE_SIZE 32
8
9  int main(){
10     int child_1, child_2;
11
12     //initializing pipe
13     int fd[2];
14     if (pipe(fd) == -1){
15         printf("Coundn't create a pipe\n");
16         exit(1);
17     }
18
19     child_1 = fork();
20     if(child_1 == -1){
21         perror("Coulnd't fork child #1");
22         exit(1);
23     }
24     if (child_1 == 0){
25
26
27         close(fd[0]);
28         if (write(fd[1], "Hello from child #1, parent!\n",
29             MESSAGE_SIZE) > 0)
30             printf("Child #1 sent a greeting to the parent\n");
31         return 0;
32     }
33     if (child_1 > 0){
34         child_2 = fork();
35         if(child_2 == -1){
36             perror("Coulnd't fork child #2");
37             exit(1);
38         }
39         if (child_2 == 0){
40             sleep(1);
41             close(fd[0]);
42             if (write(fd[1], "Hello from child #2, parent!\n",
43                 MESSAGE_SIZE) > 0)
44                 printf("Child #2 sent a greeting to the parent\n");
45             return 0;
46         }else{
47
48             char msg1[MESSAGE_SIZE], msg2[MESSAGE_SIZE];
```

```

49     close(fd[1]);
50     read(fd[0], msg1, MESSAGE_SIZE);
51     read(fd[0], msg2, MESSAGE_SIZE);
52
53     printf("Parent read a message from his children: \n%s\n%s",
54           msg1, msg2);
55
56
57
58     pid_t child_pid;
59     int status;
60
61     //waiting for the second child to finish
62     child_pid = wait(&status);
63     if (WIFEXITED(status))
64         printf("\nParent: child with pid = %d finished with code %d\n",
65               child_pid, WEXITSTATUS(status) );
66     else if (WIFSTOPPED(status))
67         printf("\nParent: child %d finished with code %d\n",
68               child_pid, WSTOPSIG(status) );
69
70     //waiting for the first child to finish
71     child_pid = wait(&status);
72     if (WIFEXITED(status))
73         printf("\nParent: child with pid = %d finished with code %d\n",
74               child_pid, WEXITSTATUS(status) );
75     else if (WIFSTOPPED(status))
76         printf("\nParent: child %d finished with code %d\n",
77               child_pid, WSTOPSIG(status) );
78
79     return 0;
80 }
81 }
82 }

```

```

1  /*
2  DIY sig handler
3  */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <signal.h>
8
9  #define MESSAGE_SIZE 32
10
11 void mySignalHandler(int snum){
12     printf("\n\nHandlig signal snum = %d in prosses...\n", snum);
13     printf("Done!\n\n");
14 }
15
16 int main(){
17     int child_1, child_2;
18     signal(SIGINT, mySignalHandler);
19
20     //initializing pipe
21     int fd[2];
22     if (pipe(fd) == -1){
23         printf("Coundn't create a pipe\n");
24         exit(1);
25     }
26
27     child_1 = fork();
28     if(child_1 == -1){
29         perror("Coulnd't fork child #1");
30         exit(1);
31     }
32     if (child_1 == 0){
33         close(fd[0]);
34         if (write(fd[1], "Hello from child #1, parent!\n",
35             MESSAGE_SIZE) > 0)
36             printf("Child #1 sent a greeting to the parent\n");
37         return 0;
38     }
39     if (child_1 > 0){
40         child_2 = fork();
41         if(child_2 == -1){
42             perror("Coulnd't fork child #2");
43             exit(1);
44         }
45         if (child_2 == 0){
46             close(fd[0]);
47             if (write(fd[1], "Hello from child #2, parent!\n",
48                 MESSAGE_SIZE) > 0)

```

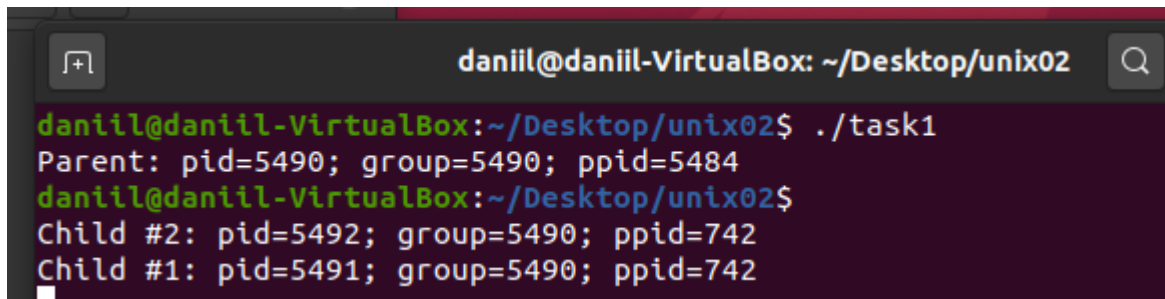
```

49     printf("Child #2 sent a greeting to the parent\n");
50     return 0;
51     }else{
52         printf("\nParent's waiting for Ctrl+C being pressed
53             to read messages from children\n");
54     pause();
55
56     char msg1[MESSAGE_SIZE], msg2[MESSAGE_SIZE];
57     close(fd[1]);
58     read(fd[0], msg1, MESSAGE_SIZE);
59     read(fd[0], msg2, MESSAGE_SIZE);
60
61     printf("Parent read this from his children: \n%s\n%s",
62         msg1, msg2);
63
64     pid_t child_pid;
65     int status;
66
67     //waiting for the second child to finish
68     child_pid = wait(&status);
69     if (WIFEXITED(status))
70         printf("\nParent: child with pid = %d finished with code %d\n",
71             child_pid, WEXITSTATUS(status) );
72     else if (WIFSTOPPED(status))
73         printf("\nParent: child %d finished with code %d\n",
74             child_pid, WSTOPSIG(status) );
75
76     //waiting for the first child to finish
77     child_pid = wait(&status);
78     if (WIFEXITED(status))
79         printf("\nParent: child with pid = %d finished with code %d\n",
80             child_pid, WEXITSTATUS(status) );
81     else if (WIFSTOPPED(status))
82         printf("\nParent: child %d finished with code %d\n",
83             child_pid, WSTOPSIG(status) );
84
85     return 0;
86     }
87 }
88 }

```

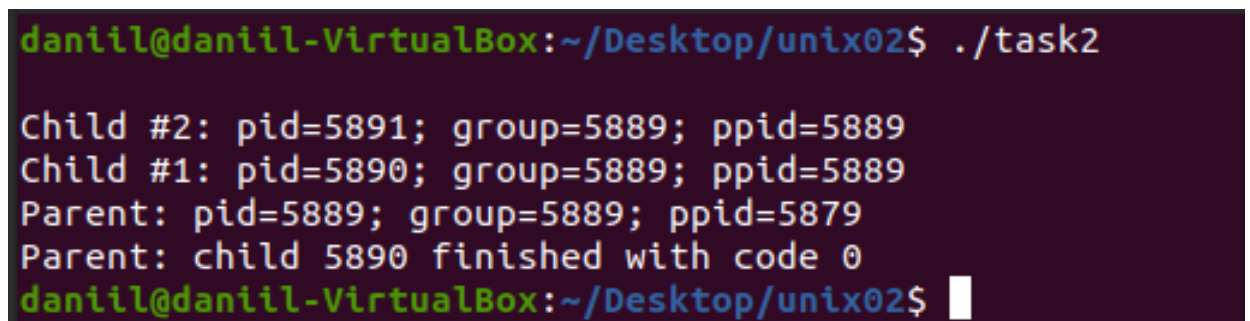
2.2 Примеры работы программ

В данном разделе будут приведены скриншоты работы программ.



```
daniil@daniil-VirtualBox: ~/Desktop/unix02
daniil@daniil-VirtualBox:~/Desktop/unix02$ ./task1
Parent: pid=5490; group=5490; ppid=5484
daniil@daniil-VirtualBox:~/Desktop/unix02$
Child #2: pid=5492; group=5490; ppid=742
Child #1: pid=5491; group=5490; ppid=742
```

Рис. 2.1: Пример работы программы №1



```
daniil@daniil-VirtualBox:~/Desktop/unix02$ ./task2
Child #2: pid=5891; group=5889; ppid=5889
Child #1: pid=5890; group=5889; ppid=5889
Parent: pid=5889; group=5889; ppid=5879
Parent: child 5890 finished with code 0
daniil@daniil-VirtualBox:~/Desktop/unix02$
```

Рис. 2.2: Пример работы программы №2

```

danil@danil-VirtualBox:~/Desktop/unix02$ ./task3

Parent: pid=5957; group=5957; ppid=5879

Child #2 executes ls -l

total 120
-rwxrwxr-x 1 danil danil 17000 ноя  8 18:26 task1
-rw-rw-r-- 1 danil danil   734 ноя  8 18:28 task1.c
-rwxrwxr-x 1 danil danil 17096 ноя  9 15:52 task2
-rw-rw-r-- 1 danil danil  1121 ноя  9 16:01 task2.c
-rwxrwxr-x 1 danil danil 17184 ноя  9 16:29 task3
-rw-rw-r-- 1 danil danil  1633 ноя  9 16:29 task3.c
-rwxrwxr-x 1 danil danil 17176 ноя  9 17:06 task4
-rw-rw-r-- 1 danil danil  1903 ноя  9 17:17 task4.c
-rwxrwxr-x 1 danil danil 17256 ноя 10 18:39 task5
-rw-rw-r-- 1 danil danil  2132 ноя 10 18:40 task5.c
.
| Parent: child with pid = 5959 finished with code 0

Child #1 executes ps -al
ng-
  F S   UID     PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
era4 S   1000      830    822   0  80   0 - 76634 ep_pol tty2        00:00:28 Xorg
0 S   1000      942    822   0  80   0 - 47711 poll_s tty2        00:00:00 gnome-ses
0 S   1000     5957    5879   0  80   0 -   622 do_wai pts/0        00:00:00 task3
4 R   1000     5958    5957   0  80   0 -  2850 -      pts/0        00:00:00 ps

Parent: child with pid = 5958 finished with code 0
danil@danil-VirtualBox:~/Desktop/unix02$ █

```

Рис. 2.3: Пример работы программы №3

```

danil@danil-VirtualBox:~/Desktop/unix02$ ./task4
Child #1 sent a greeting to the parent
Child #2 sent a greeting to the parent
Parent read a message from his children:
Hello from child #1, parent!

Hello from child #2, parent!

Parent: child with pid = 6000 finished with code 0

Parent: child with pid = 6001 finished with code 0
danil@danil-VirtualBox:~/Desktop/unix02$ █

```

Рис. 2.4: Пример работы программы №4

```
Parent: child with pid = 6041 finished with code 0
daniil@danil-VirtualBox:~/Desktop/unix02$ ./task5
|
| Parent's waiting for Ctrl+C being pressed to read messages from children
dcChild #2 sent a greeting to the parent
g-Child #1 sent a greeting to the parent
^C
r2
Handling signal snum = 2 in proses...
Done!

Parent read a message from his children:
Hello from child #2, parent!

Hello from child #1, parent!

Parent: child with pid = 6041 finished with code 0

Parent: child with pid = 6042 finished with code 0
```

Рис. 2.5: Пример работы программы №5

Вывод

В данном разделе были рассмотрены листинг кода.

Заключение

В ходе лабораторной работы были изучены системные вызовы `exec()`, `fork()` и реализованы алгоритмы, демонстрирующие работу этих вызовов.