

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №4

Дисциплина: Операционные системы

(И.О. Фамилия)

(И.О. Фамилия)

Москва, 2021

Задание №1

Задание: 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.

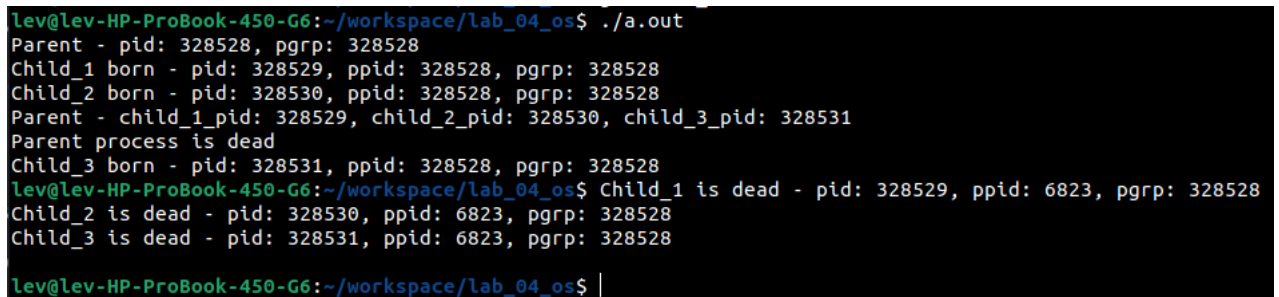
```
1 #include <unistd.h>
2 #include <stdio.h>
3
4 #define FORK_ERROR -1
5 #define FORK_OK 0
6
7 #define COUNT_CHILDS 3
8
9 int main(void)
10 {
11     pid_t chlds_pid[COUNT_CHILDS];
12
13     printf("Parent - pid: %d, pgrp: %d\n", getpid(), getpgrp());
14
15     for (int i = 0; i < COUNT_CHILDS; i++)
16     {
17         pid_t child_pid = fork();
18         if (child_pid == FORK_ERROR)
19         {
20             perror("Can't fork.\n");
21             return 1;
22         }
23         else if (child_pid == FORK_OK)
24         {
25             printf("Child_%d born - pid: %d, ppid: %d, pgrp: %d\n",
26                   i + 1, getpid(), getppid(), getpgrp());
27             sleep(2);
28             printf("Child_%d is dead - pid: %d, ppid: %d, pgrp: %d\n",
29                   i + 1, getpid(), getppid(), getpgrp());
30             return 0;
31         }
32         else{
33             chlds_pid[i] = child_pid;
```

```

34     }
35
36 }
37
38 printf("Parent-child_1_pid: %d, child_2_pid: %d, child_3_pid: %d\n",
39        childs_pid[0], childs_pid[1], childs_pid[2]);
40 printf("Parent process is dead.\n");
41
42 return 0;
43 }

```

Результат работы программы представлен на рисунке 1:



```

lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ ./a.out
Parent - pid: 328528, pgrp: 328528
Child_1 born - pid: 328529, ppid: 328528, pgrp: 328528
Child_2 born - pid: 328530, ppid: 328528, pgrp: 328528
Parent - child_1_pid: 328529, child_2_pid: 328530, child_3_pid: 328531
Parent process is dead
Child_3 born - pid: 328531, ppid: 328528, pgrp: 328528
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ Child_1 is dead - pid: 328529, ppid: 6823, pgrp: 328528
Child_2 is dead - pid: 328530, ppid: 6823, pgrp: 328528
Child_3 is dead - pid: 328531, ppid: 6823, pgrp: 328528
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ |

```

Рисунок 1 – Демонстрация работы программы (задание 1).

Задание №2

Задание: 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 – Использование системного вызова `wait()`.

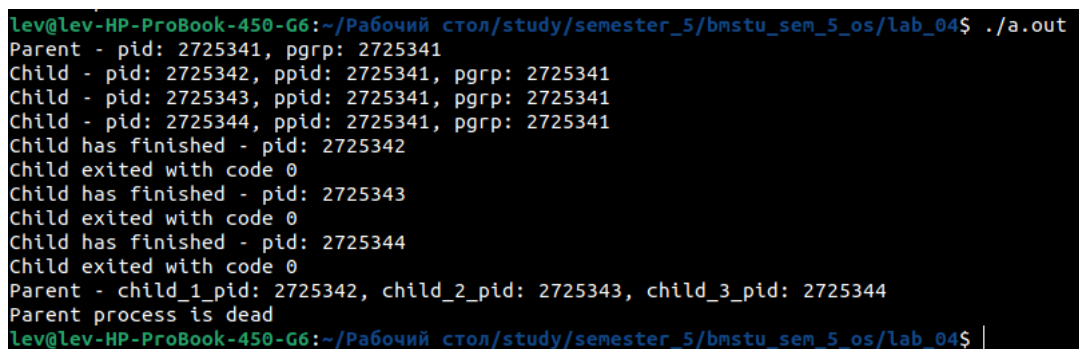
```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <sys/wait.h>
4
5 #define FORK_ERROR -1
6 #define FORK_OK 0
7
8 #define COUNT_CHILDS 3
9
10 int main(void)
11 {
12     pid_t chlds_pid[COUNT_CHILDS];
13     pid_t child_pid;
14
15     printf("Parent - pid: %d, pgrp: %d\n", getpid(), getpgrp());
16
17     for (int i = 0; i < COUNT_CHILDS; i++)
18     {
19         child_pid = fork();
20         if (child_pid == FORK_ERROR)
21         {
22             perror("Can't fork.\n");
23             return 1;
24         }
25         else if (child_pid == FORK_OK)
26         {
27             sleep(2);
28             printf("Child - pid: %d, ppid: %d, pgrp: %d\n",
29                   getpid(), getppid(), getpgrp());
30             return 0;
31         }
32         else{
33             chlds_pid[i] = child_pid;
34         }
35     }
36 }
37
```

```

38     for (int i = 0; i < COUNT_CHILDS; i++)
39     {
40         int status;
41         pid_t child_pid;
42         child_pid = wait(&status);
43
44         printf("Child has finished - pid: %d\n", child_pid);
45
46         int stat_val;
47         if (WIFEXITED(stat_val)){
48             printf("Child exited with code %d\n", WEXITSTATUS(stat_val));
49         }
50         else {
51             printf("Child terminated abnormally.\n");
52         }
53     }
54
55     printf("Parent-child_1_pid: %d, child_2_pid: %d, child_3_pid: %d\n",
56           childs_pid[0], childs_pid[1], childs_pid[2]);
57     printf("Parent process is dead\n");
58
59     return 0;
60 }

```

Результат работы программы представлен на рисунке 2:



```

lev@lev-HP-ProBook-450-G6:~/Рабочий стол/study/semester_5/bmstu_sem_5_os/lab_04$ ./a.out
Parent - pid: 2725341, pgrp: 2725341
Child - pid: 2725342, ppid: 2725341, pgrp: 2725341
Child - pid: 2725343, ppid: 2725341, pgrp: 2725341
Child - pid: 2725344, ppid: 2725341, pgrp: 2725341
Child has finished - pid: 2725342
Child exited with code 0
Child has finished - pid: 2725343
Child exited with code 0
Child has finished - pid: 2725344
Child exited with code 0
Parent - child_1_pid: 2725342, child_2_pid: 2725343, child_3_pid: 2725344
Parent process is dead
lev@lev-HP-ProBook-450-G6:~/Рабочий стол/study/semester_5/bmstu_sem_5_os/lab_04$ |

```

Рисунок 2 – Демонстрация работы программы (задание 2).

Задание №3

Задание: 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.

В данном задании системному вызову `exec()` были переданы исполняемые файлы "main_1.out" и "main_2.out". Программа main_1.out соответствует программной реализации алгоритма полиномиальной интерполяции табличных функций. Программа main_2.out соответствует программной реализации алгоритма сплайн-интерполяции табличных функций.

Листинг 3 – Использование системного вызова `exec()`.

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <sys/wait.h>
4
5 #define FORK_ERROR -1
6 #define FORK_OK 0
7
8 #define COUNT_CHILDS 2
9
10 int main(void)
11 {
12     pid_t childs_pid[COUNT_CHILDS];
13     pid_t child_pid;
14
15     char *command_args[COUNT_CHILDS] = {"/home/lev/workspace/lab_04_os/
        main_1.out", "/home/lev/workspace/lab_04_os/main_2.out"};
16
17     int res_exec = 0;
18
19     printf("Parent - pid: %d, pgrp: %d\n", getpid(), getpgrp());
20
21     for (int i = 0; i < COUNT_CHILDS; i++)
22     {
23         child_pid = fork();
24         if (child_pid == FORK_ERROR)
25         {
26             perror("Can't fork.\n");
27             return 1;
```

```

28     }
29     else if (child_pid == FORK_OK)
30     {
31         printf("Child - pid: %d, ppid: %d, pgrp: %d\n", getpid(),
32             getppid(), getpgrp());
33         if (i == 0){
34             res_exec = execlp(command_args[i], "main_1.out", "
35                 table_1_newton.txt",
36                     (char *)NULL);
37             }
38             else{
39                 res_exec = execlp(command_args[i], command_args[i], (
40                     char *)NULL);
41             }
42             if (res_exec == -1){
43                 perror("Can't child exec(). ");
44             }
45             return 0;
46         }
47         else{
48             childs_pid[i] = child_pid;
49         }
50     }
51     for (int i = 0; i < COUNT_CHILDS; i++)
52     {
53         int status;
54         pid_t child_pid;
55         child_pid = wait(&status);
56
57         printf("Child has finished - pid: %d, ppid = %d\n", child_pid,
58             getppid());
59
60         if (WIFEXITED(status)){
61             printf("Child exited with code %d\n", WEXITSTATUS(status));
62         }
63         else {
64             printf("Child terminated abnormally.\n");
65         }
66     }
67
68     printf("Parent-child_1_pid: %d, child_2_pid: %d, child_3_pid: %d\n",
69         childs_pid[0], childs_pid[1], childs_pid[2]);
70     printf("Parent process is dead\n");
71
72     return 0;

```

Результат работы программы представлен на рисунке 3:

```
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ ./a.out
Parent - pid: 94587, pgrp: 94587
Child - pid: 94588, ppid: 94587, pgrp: 94587
Child - pid: 94589, ppid: 94587, pgrp: 94587
Введите значение аргумента, для которого выполняется интерполяция: Введите количество узлов: 0.525
Введенный аргумент = 0.525000
Результирующая таблица:
| Степень полинома | Полином Ньютона | Полином Эрмита |
|-----|-----|-----|
| 0 | 0.450447 | 0.342824 |
| 1 | 0.337891 | 0.342824 |
| 2 | 0.340419 | 0.340323 |
| 3 | 0.340314 | 0.340323 |
| 4 | 0.340324 | 0.340427 |
res-newton = 0.340314
Child has finished - pid: 94588, ppid = 58918
Child exited with code 0
11
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
Введите аргумент полинома: 3.5
Результат = 12.252830
Child has finished - pid: 94589, ppid = 58918
Child exited with code 0
Parent - child_1_pid: 94588, child_2_pid: 94589
Parent process is dead
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ |
```

Рисунок 3 – Демонстрация работы программы (задание 3).

Задание №4

Задание: 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 – Обмен сообщениями предка и потомка через неименованный программный канал.

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <sys/wait.h>
4 #include <string.h>
5
6 #define FORK_ERROR -1
7 #define FORK_OK 0
8 #define BUFFER_SIZE 256
9
10 #define COUNT_CHILDS 3
11
12 int main(void)
13 {
14     pid_t chlds_pid[COUNT_CHILDS];
15     pid_t child_pid;
16
17     int file_descriptors[2];
18     const char *messages[COUNT_CHILDS] = {"gegwege\n", "
19     fwgrzdjnkfjrbgegb\n", "ng\n"};
20     char buffer[BUFFER_SIZE] = {0};
21
22     int res_exec = 0;
23
24     pipe(file_descriptors);
25
26     printf("Parent - pid: %d, pgrp: %d\n", getpid(), getpgrp());
27
28     for (int i = 0; i < COUNT_CHILDS; i++)
29     {
30         child_pid = fork();
31         if (child_pid == FORK_ERROR)
```

```

32         perror("Can't fork.\n");
33         return 1;
34     }
35     else if (child_pid == FORK_OK)
36     {
37         close(file_descriptors[0]);
38         write(file_descriptors[1], messages[i], strlen(messages[i]))
39         ;
40         printf("Message from child_%d sent to the parent.\n", i + 1)
41         ;
42         return 0;
43     }
44     else{
45         childs_pid[i] = child_pid;
46     }
47 }
48
49 for (int i = 0; i < COUNT_CHILDS; i++)
50 {
51     int status;
52     pid_t child_pid = wait(&status);
53
54     printf("Child has finished - pid: %d, ppid = %d\n", child_pid,
55           getppid());
56
57     if (WIFEXITED(status)){
58         printf("Child exited with code %d\n", WEXITSTATUS(status));
59     }
60     else {
61         printf("Child terminated abnormally.\n");
62     }
63 }
64
65 close(file_descriptors[1]);
66 read(file_descriptors[0], buffer, BUFFER_SIZE);
67
68 printf("Received message:\n%s", buffer);
69
70 printf("Parent - child_1_pid: %d, child_2_pid: %d, child_3_pid: %d\n
71        ", childs_pid[0], childs_pid[1], childs_pid[2]);
72 printf("Parent process is dead\n");
73
74 return 0;
75 }

```

Результат работы программы представлен на рисунке 4:

```
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ ./a.out
Parent - pid: 451885, pgrp: 451885
Message from child_1 sent to the parent.
Message from child_2 sent to the parent.
Message from child_3 sent to the parent.
Child has finished - pid: 451886, ppid = 3532379
Child exited with code 0
Child has finished - pid: 451887, ppid = 3532379
Child exited with code 0
Child has finished - pid: 451888, ppid = 3532379
Child exited with code 0
Received message:
gegwege
fwgrzdjnkfjrbgegb
ng
Parent - child_1_pid: 451886, child_2_pid: 451887, child_3_pid: 451888
Parent process is dead
```

Рисунок 4 – Демонстрация работы программы (задание 4).

Задание №5

Задание: предок и потомки аналогично п.4 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 – Использование сигнала.

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <sys/wait.h>
4 #include <string.h>
5 #include <signal.h>
6
7 #define FORK_ERROR -1
8 #define FORK_OK 0
9 #define BUFFER_SIZE 256
10
11 #define COUNT_CHILDS 3
12 #define NO_SIGNAL 0
13 #define GET_SIGNAL 1
14
15 int mode = NO_SIGNAL;
16 void change_mode(int signal_number){
17     mode = GET_SIGNAL;
18 }
19
20 int main(void)
21 {
22     pid_t childs_pid[COUNT_CHILDS];
23     pid_t child_pid;
24
25     int file_descriptors[2];
26     const char *messages[COUNT_CHILDS] = {"gegwege\n", "
27     fwgrzdjnkfjrbgegb\n", "ng\n"};
28     char buffer[BUFFER_SIZE] = {0};
29
30     int res_exec = 0;
31
32     pipe(file_descriptors);
33     signal(SIGINT, change_mode);
```

```

33
34     printf("Parent - pid: %d, pgrp: %d\n", getpid(), getpgrp());
35
36     for (int i = 0; i < COUNT_CHILDS; i++)
37     {
38         child_pid = fork();
39         if (child_pid == FORK_ERROR)
40         {
41             perror("Can't fork.\n");
42             return 1;
43         }
44         else if (child_pid == FORK_OK)
45         {
46             sleep(2);
47             if (mode == GET_SIGNAL)
48             {
49                 close(file_descriptors[0]);
50                 write(file_descriptors[1], messages[i], strlen(messages[
51                     i]));
52                 printf("Message from child_%d sent to the parent.\n", i
53                     + 1);
54             }
55             else{
56                 printf("No signal sent!\n");
57             }
58
59             return 0;
60         }
61         else{
62             childs_pid[i] = child_pid;
63         }
64     }
65
66     for (int i = 0; i < COUNT_CHILDS; i++)
67     {
68         int status = 0;
69         child_pid = wait(&status);
70
71         printf("Child has finished - pid: %d, ppid = %d\n", child_pid,
72             getpid());
73
74         if (WIFEXITED(status)){
75             printf("Child exited with code %d\n", WEXITSTATUS(status));
76         }
77         else {
78             printf("Child terminated abnormally.\n");
79         }
80     }

```

```

78     }
79
80
81     close(file_descriptors[1]);
82     read(file_descriptors[0], buffer, BUFFER_SIZE);
83
84     printf("Received message:\n%s", buffer);
85
86     printf("Parent - child_1_pid: %d, child_2_pid: %d, child_3_pid: %d\n",
87           childs_pid[0], childs_pid[1], childs_pid[2]);
88     printf("Parent process is dead\n");
89
90     return 0;
91 }

```

Результат работы программы в случае, когда сигнал не поступает, представлен на рисунке 5:

```

lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ ./a.out
Parent - pid: 2703778, pgrp: 2703778
No signal sent!
No signal sent!
No signal sent!
Child has finished - pid: 2703779, ppid = 3999096
Child exited with code 0
Child has finished - pid: 2703780, ppid = 3999096
Child exited with code 0
Child has finished - pid: 2703781, ppid = 3999096
Child exited with code 0
Received message:
Parent - child_1_pid: 2703779, child_2_pid: 2703780, child_3_pid: 2703781
Parent process is dead
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ |

```

Рисунок 5 – Демонстрация работы программы, сигнал не поступает (задание 5).

Результат работы программы в случае, когда сигнал поступил, представлен на рисунке 6:

```

lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ ./a.out
Parent - pid: 2682687, pgrp: 2682687
^CMessage from child_3 sent to the parent.
Message from child_2 sent to the parent.
Message from child_1 sent to the parent.
Child has finished - pid: 2682690, ppid = 3999096
Child exited with code 0
Child has finished - pid: 2682688, ppid = 3999096
Child exited with code 0
Child has finished - pid: 2682689, ppid = 3999096
Child exited with code 0
Received message:
ng
fwgrzdjnkfjrbgegb
gegwege
Parent - child_1_pid: 2682688, child_2_pid: 2682689, child_3_pid: 2682690
Parent process is dead
lev@lev-HP-ProBook-450-G6:~/workspace/lab_04_os$ gcc task_5.c

```

Рисунок 6 – Демонстрация работы программы, сигнал поступил (задание 5).