



КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

Преподаватели Рязанова Н. Ю.

## Задание №1

Процессы-сироты. В программе создаются не менее двух потомков. В потомках вызывается `sleep()`. Чтобы предок гарантированно завершился раньше своих помков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе.

Листинг 1: Процессы-сироты

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     int childpid_first = fork(), childpid_second;
7
8     if (childpid_first == -1)
9     {
10         perror("Can't fork\n");
11         return -1;
12     }
13     else if (childpid_first == 0)
14     {
15         sleep(1);
16         printf("\nFirst child: id: %d ppid: %d pgroup: %d\n", getpid(),
17             getppid(), getpgrp());
18         return 0;
19     }
20
21     childpid_second = fork();
22     if (childpid_second == -1)
23     {
24         perror("Can't fork\n");
25         return -1;
26     }
27     else if (childpid_second == 0)
28     {
29         sleep(1);
30         printf("\nSecond child: id: %d ppid: %d pgroup: %d\n\n", getpid(),
31             getppid(), getpgrp());
32         return 0;
33     }
34
35     printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n",
36         getpid(), getpgrp(), childpid_first, childpid_second);
37
38     return 0;
39 }
```

```
ilyaartemev@Air-Ilya > ~/Desktop/IU7-0S/lab_04/src > master ● ? > ./case_1.exe
Parent: id: 16397 pgroup: 16397 first child: 16398 second child: 16399
ilyaartemev@Air-Ilya > ~/Desktop/IU7-0S/lab_04/src > master ● ?
First child: id: 16398 ppid: 1 pgroup: 16397

Second child: id: 16399 ppid: 1 pgroup: 16397

ilyaartemev@Air-Ilya > ~/Desktop/IU7-0S/lab_04/src > master ● ?
```

Рис. 1: Демонстрация работы программы (задание №1).

## Задание №2

Предок ждет завершения своих потомков, используя системный вызов `wait()`. Вывод соответствующих сообщений на экран.

Листинг 2: Вызов функции `wait()`

```
1 #include "status.h"
2
3 int main(void)
4 {
5     int status;
6     int childpid_first = fork(), childpid_second;
7
8     if (childpid_first == -1)
9     {
10         perror("Can't fork\n");
11         return -1;
12     }
13     else if (childpid_first == 0)
14     {
15
16         printf("\nFirst child: id: %d ppid: %d pgroup: %d\n", getpid(),
17             getppid(), getpgrp());
18         return 0;
19     }
20
21     childpid_second = fork();
22     if (childpid_second == -1)
23     {
24         perror("Can't fork\n");
25         return -1;
26     }
27     else if (childpid_second == 0)
28     {
29
30         printf("\nSecond child: id: %d ppid: %d pgroup: %d\n\n", getpid(),
31             getppid(), getpgrp());
32         return 0;
33     }
34
35     wait(&status);
36     check_status(status);
37     wait(&status);
38     check_status(status);
39
40     printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n",
41         getpid(), getpgrp(), childpid_first, childpid_second);
42
43     return 0;
44 }
```

```
ilyaartemev@Air-Ilya ~/Desktop/IU7-OS/lab_04/src master ● ? ./case_2.exe  
First child: id: 16763 ppid: 16762 pgroup: 16762  
Дочерний процесс завершен нормально  
Код завершения дочернего процесса: 0  
Second child: id: 16764 ppid: 16762 pgroup: 16762  
Дочерний процесс завершен нормально  
Код завершения дочернего процесса: 0  
Parent: id: 16762 pgroup: 16762 first child: 16763 second child: 16764  
ilyaartemev@Air-Ilya ~/Desktop/IU7-OS/lab_04/src master ● ?
```

Рис. 2: Демонстрация работы программы (задание №2).

## Задание №3

Потомки переходят на выполнение других программ. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 3: Вызов функции `execvp()`

```
1 #include "status.h"
2
3 int main(void)
4 {
5     int status;
6     int childpid_first = fork(), childpid_second;
7
8     if (childpid_first == -1)
9     {
10         perror("Can't fork\n");
11         return -1;
12     }
13     else if (childpid_first == 0)
14     {
15         printf("\nFirst child: id: %d ppid: %d pgroup: %d\n", getpid(),
16             getppid(), getpgrp());
17
18         if (execvp("./levenstein.exe", "./levenstein.exe", NULL) == -1)
19         {
20             perror("Can't execvp first child\n\n");
21             return -1;
22         }
23
24         return 0;
25     }
26
27     childpid_second = fork();
28     if (childpid_second == -1)
29     {
30         perror("Can't fork\n");
31         return -1;
32     }
33     else if (childpid_second == 0)
34     {
35         printf("\nSecond child: id: %d ppid: %d pgroup: %d\n\n", getpid(),
36             getppid(), getpgrp());
37
38         sleep(8);
39         if (execvp("./sort.exe", "./sort.exe", NULL) == -1)
40         {
41             perror("Can't execvp second child\n\n");
42             return -1;
43         }
44     }
```

```

43     return 0;
44 }
45
46 wait(&status);
47 check_status(status);
48 wait(&status);
49 check_status(status);
50
51 printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n",
52        getpid(), getpgrp(), childpid_first, childpid_second);
53
54 return 0;
55 }

```

```

ilyaartemev@Air-Ilya ~/Desktop/IU7-0S/lab_04/src master ● ? ./case_3.exe
First child: id: 16930 ppid: 16929 pgroup: 16929
Second child: id: 16931 ppid: 16929 pgroup: 16929
Введите первую строку: aba
Введите вторую строку: ba
Редакционное расстояние: 1
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Введите количество элементов массива: 4
Введите элементы массива: 0 4 1 -43
Отсортированный массив: -43 0 1 4
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Parent: id: 16929 pgroup: 16929 first child: 16930 second child: 16931
ilyaartemev@Air-Ilya ~/Desktop/IU7-0S/lab_04/src master ● ?

```

Рис. 3: Демонстрация работы программы (задание №3).

## Задание №4

Предок и потомки обмениваются сообщениями через неименованный программный канал. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 4: Использование pipe

```
1 #include "status.h"
2
3 int main(void)
4 {
5     const char *msg_1 = "Overleaf is used by over eight million students and
6         academics at 3,600 institutions worldwide.\n";
7     const char *msg_2 = "Just want to say that I am really grateful for
8         Overleaf, it has enabled a slew of research and teaching development
9         in my work that would have been annoyingly difficult before. Even
10        people who don't know LaTeX are participating with me on research
11        proposals and that's saying something if you know what LaTeX is
12        like for the uninitiated.\n";
13
14     char readbuffer[1000];
15     int fd[2];
16
17     int status;
18     int childpid_first, childpid_second;
19
20     if (pipe(fd) == -1)
21     {
22         printf("1\n");
23         perror("Can't pipe\n");
24         return -1;
25     }
26
27     childpid_first = fork();
28     if (childpid_first == -1)
29     {
30         perror("Can't fork.\n");
31         return -1;
32     }
33     else if (childpid_first == 0)
34     {
35         close(fd[0]);
36         write(fd[1], msg_1, strlen(msg_1));
37
38         return 0;
39     }
40
41     childpid_second = fork();
42     if (childpid_second == -1)
43     {
44         perror("Can't fork\n");
45     }
```



```

39     return -1;
40 }
41 else if (childpid_second == 0)
42 {
43     close(fd[0]);
44     write(fd[1], msg_2, strlen(msg_2));
45
46     return 0;
47 }
48
49 wait(&status);
50 check_status(status);
51 wait(&status);
52 check_status(status);
53
54 printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n\n",
55        getpid(), getppgrp(), childpid_first, childpid_second);
56
57 close(fd[1]);
58 read(fd[0], readbuffer, sizeof(readbuffer));
59 printf("%s", readbuffer);
60
61 return 0;
62 }

```

```

ilyaartemev@Air-Ilya ~/Desktop/IU7-0S/lab_04/src master • ? ./case_4.exe 11688 16:59:50
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Parent: id: 17575 pgroup: 17575 first child: 17576 second child: 17577
Overleaf is used by over eight million students and academics at 3,600 institutions worldwide.
Just want to say that I am really grateful for Overleaf, it has enabled a slew of research and teaching development
in my work that would have been annoyingly difficult before. Even people who don't know LaTeX are participating wi
th me on research proposals and that's saying something if you know what LaTeX is like for the uninitiated.
ilyaartemev@Air-Ilya ~/Desktop/IU7-0S/lab_04/src master • ? 11689 16:59:52

```

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

## Задание №5

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.

Листинг 5: Использование сигналов

```
1 #include "status.h"
2
3 int flag = 0;
4
5 void ctrl_c(const int signal_numb)
6 {
7     flag = 1;
8     printf("\nSignal number: %d\n", signal_numb);
9 }
10
11 int main(void)
12 {
13     const char *msg_1 = "Overleaf is used by over eight million students and
14     academics at 3,600 institutions worldwide.\n";
15     const char *msg_2 = "Just want to say that I am really grateful for
16     Overleaf, it has enabled a slew of research and teaching development
17     in my work that would have been annoyingly difficult before. Even
18     people who dont know LaTeX are participating with me on research
19     proposals and that's saying something if you know what LaTeX is
20     like for the uninitiated.\n";
21
22     char readbuffer[1000];
23     int fd[2];
24
25     int status;
26     int childpid_first, childpid_second;
27
28     signal(SIGINT, ctrl_c);
29
30     if (pipe(fd) == -1)
31     {
32         perror("Can't pipe\n");
33         return -1;
34     }
35
36     printf("If you want to write messages to the channel, press: \nCTRL+C
37     \n.\n");
38     sleep(20);
39
40     childpid_first = fork();
41     if (childpid_first == -1)
42     {
43         perror("C a n t fork.\n");
```

```

37     return -1;
38 }
39 else if (childpid_first == 0)
40 {
41     if (flag == 1)
42     {
43         close(fd[0]);
44         write(fd[1], msg_1, strlen(msg_1));
45     }
46
47     return 0;
48 }
49
50 childpid_second = fork();
51 if (childpid_second == -1)
52 {
53     perror("Can't fork\n");
54     return -1;
55 }
56 else if (childpid_second == 0)
57 {
58     if (flag == 1)
59     {
60         close(fd[0]);
61         write(fd[1], msg_2, strlen(msg_2));
62     }
63
64     return 0;
65 }
66
67 wait(&status);
68 check_status(status);
69 wait(&status);
70 check_status(status);
71
72 printf("Parent: id: %d pgroup: %d first child: %d second child: %d\n\n",
73        getpid(), getpgrp(), childpid_first, childpid_second);
74
75 if (flag == 1)
76 {
77     close(fd[1]);
78     read(fd[0], readbuffer, sizeof(readbuffer));
79     printf("%s", readbuffer);
80 }
81
82 return 0;
83 }

```

```
ilyaartemev@Air-Ilya > ~/Desktop/IU7-05/lab_04/src master ● ? > ./case_5.exe 10263 15:16:48
If you want to write messages to the channel, press: "CTRL+C".
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Parent: id: 74437 pgroup: 74437 first child: 74451 second child: 74452
ilyaartemev@Air-Ilya > ~/Desktop/IU7-05/lab_04/src master ● ? > 10263 15:17:22
```

Рис. 5: Демонстрация работы программы, сигнал не вызывается (задание №5).

```
ilyaartemev@Air-Ilya > ~/Desktop/IU7-05/lab_04/src master ● ? > ./case_5.exe 10263 15:16:37
If you want to write messages to the channel, press: "CTRL+C".
^C
Signal №: 2
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Дочерний процесс завершен нормально
Код завершения дочернего процесса: 0
Parent: id: 74324 pgroup: 74324 first child: 74325 second child: 74326
Overleaf is used by over eight million students and academics at 3,600 institutions worldwide.
Just want to say that I am really grateful for Overleaf, it has enabled a slew of research and teaching development
in my work that would have been annoyingly difficult before. Even people who don't know LaTeX are participating wi
th me on research proposals and that's saying something if you know what LaTeX is like for the uninitiated.
```

Рис. 6: Демонстрация работы программы, сигнал вызывается (задание №5).

## Дополнительные файлы

Листинг 6: Проверка статуса завершившегося дочернего процесса

```
1 #include "status.h"
2
3 void check_status(const int status)
4 {
5     if (WIFEXITED(status))
6     {
7         printf("Child process completed successfully\n\n");
8         printf("Return code: %d\n\n", WEXITSTATUS(status));
9     }
10    else if (WIFSIGNALED(status))
11    {
12        printf("The child process terminates with an un-intercepted signal\n\n");
13        printf("Signal number: %d\n\n", WTERMSIG(status));
14    }
15    else if (WIFSTOPPED(status))
16    {
17        printf("The child process has stopped\n\n");
18        printf("Signal number: %d\n\n", WSTOPSIG(status));
19    }
20
21    return;
22 }
```

Листинг 7: Хедер для подключения к файлам вызывающим check\_status

```
1 #ifndef __STATUS_H__
2 #define __STATUS_H__
3
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <sys/wait.h>
8
9 void check_status(const int status);
10
11 #endif // __STATUS_H__
```

## Дополнительное задание

- Порядок выполнения вызова `fork()`:
- 1) Резервируется пространство памяти для данных и стека процесса - потомка;
  - 2) Наблюдается идентификатор: процесс `PID` и структура `proc` потомка;
  - 3) Инициализируется структура `proc` потомка. Некоторые поля этой структуры копируются от процесса - родителя: идентификаторы наследоваться и группы, места сегментов и группа процессов. Часть полей инициализируется специальными значениями для потомка значениями: `PID` потомка и его родитель, указатель на структуру `proc` родителя;
  - 4) Создаются карты трансляции адресов для процесса - потомка;
  - 5) Выделяется область и память и в нее копируется область и процесс - родитель;
  - 6) Обновляются ссылки области и на новые карты адресации и пространство памяти;
  - 7) Потомок добавляется в табл. процессов, которые разделяют область кода программы выполняемой процессом - родителем;
  - 8) Постепенно разрушаются области данных и стека родителя и родителя.

Рис. 7: Очередность при вызове `fork()`.



- циркулирующая карта: адресация потока;
- 9) Поток получает доступ к различным ресурсам, которыми он владеет: открытым файлам (поток владеет дескрипторами) и текущей рабочей памятью;
  - 10) Инициализируется атрибутный контекст потока путем копирования регистров родителя;
  - 11) Запустить процесс - поток в очередь готовых процессов;
  - 12) Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 - в процессе - потоке.

Рис. 8: Очередность при вызове fork().

При выполнении вызова `exec()`:

- 1) Избрасывает путь к исполняемому файлу и осуществляет доступ к нему.
- 2) Проверяет, имеет ли вызывающий процесс полномочия на выполнение файла.
- 3) Читает заголовок и проверяет, что он действительно исполняемый.
- 4) Если для файла установлен бит `SUID` или `SGID`, то соответствующие идентификаторы `UID` и `GID` вызывающего процесса изменяет на `UID` и `GID`, соответствующие владельцу файла.
- 5) Копирует аргументы, передаваемые в `exec`, а также переменные среды в пространство ядра, после чего переходит к заголовку программы.
- 6) Вызывает пространство памяти для обмена данными и стека.
- 7) Выводится старое адресное пространство и связывается с ним пространство памяти. Если же процесс был создан при помощи `fork`, то происходит возврат старого адресного пространства родительскому процессу.
- 8) Выводит новые управляющие адреса

Рис. 9: Очередность при вызове `exec()`.



- для нового метода, деструктора и стека
- 9) Устанавливается новое адресное пространство. Если область текста совпадает (какой-то другой процесс уже выделяет эту же программу), то она будет совместно использоваться с этим процессом. В другом случае пространство должно инцидуализироваться из выделенной памяти. Благодаря в системе UNIX обычно реализуется на странице, что означает, что каждая страница считывается в память только по мере необходимости.
  - 10) Копирует аргументы и переменные среды обратно в новый стек процесса
  - 11) Сбрасывает все обработчики сигналов в действия, определенные по умолчанию, так как функции обработчиков сигналов не существуют в новой программе. Сигналы, которые были проигнорированы или заблокированы перед вызовом `exec`, остаются в том же состоянии.
  - 12) Инцидуализирует окружающий контекст. При этом большинство регистров сбрасывается в 0, а указатель на текущую часть памяти можно ввести программу

Рис. 10: Очередность при вызове `exec()`.