



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»

ДИСЦИПЛИНА «Операционные системы»

Лабораторная работа № 4

Тема «Процессы. Системные вызовы `fork()` и `exec()`»

Студент Шиленков А. А.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва.
2020 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.

Все задания в этой лабораторной работе проделаны на дистрибутиве Elementary OS (Ubuntu LTS 18)

Код программы

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int first_childpid, second_childpid;

    first_childpid = fork();
    if (first_childpid == -1)
    {
        perror("Can`t fork.\n");
        return 1;
    }
    else if (first_childpid == 0)
    {
        sleep(1);

        printf("\n1st child proc:\n");
        printf("\tPID: %d\n", getpid());
        printf("\tPPID: %d\n", getppid());
        printf("\tPGRP: %d\n", getpgrp());

        return 0;
    }
    else
    {
        if ((second_childpid = fork()) == -1)
        {
            perror("Can`t fork.\n");
            return 1;
        }
        else if (second_childpid == 0)
        {
            sleep(2);

            printf("2nd child proc:\n");
            printf("\tPID: %d\n", getpid());
            printf("\tPPID: %d\n", getppid());
            printf("\tPGRP: %d\n", getpgrp());

            return 0;
        }

        printf("Parent proc:\n");
        printf("\tPID: %d\n", getpid());
        printf("\tPID_FIRST: %d\n\tPID_SECOND %d\n", first_childpid, second_childpid);
        printf("\tPGRP: %d\n", getpgrp());

        return 0;
    }
}
```

Пример работы программы

В данном задании было рассмотрено усыновление процессов. На изображении 1 представлен пример работы программы.

```
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$ ./task1.out
Parent proc:
    PID: 11610
    PID_FIRST: 11611
    PID_SECOND: 11612
    PGRP: 11610
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$
1st child proc:
    PID: 11611
    PPID: 1
    PGRP: 11610
2nd child proc:
    PID: 11612
    PPID: 1
    PGRP: 11610
```

Изображение 1. – Работа первой программы

После завершения родительского процесса происходит усыновление процессов.

Сироты – дочерние процессы усыновляются другим процессом, это можно увидеть на рисунке 2. В моей системе это PID = 1 /sbin/init, он же systemd.

```
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$ sudo ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.2 225680  9320 ?        Ss   21:21   0:03 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    21:21   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   21:21   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   21:21   0:00 [rcu_par_gp]
root         8  0.0  0.0      0     0 ?        I<   21:21   0:00 [mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    21:21   0:00 [ksoftirqd/0]
```

Изображение 2. – Процесс PID = 1.

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.

Код программы

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    pid_t first_childpid, second_childpid;

    first_childpid = fork();
    if (first_childpid == -1)
    {
        perror("Can`t fork.\n");
        return 1;
    }
    else if (first_childpid == 0)
    {
        printf("\n1st child proc:\n");
        printf("\tPID: %d\n", getpid());
        printf("\tPPID: %d\n", getppid());
        printf("\tPGRP: %d\n", getpgrp());

        return 0;
    }
    else
    {
        int status;

        first_childpid = wait(&status);
        printf("1st child has finished: PID = %d\n", first_childpid);

        if (WIFEXITED(status))
            printf("1st child exited with code %d\n", WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
            printf("1st child exited with signal number %d\n", WTERMSIG(status));
        else if (WIFSTOPPED(status))
            printf("1st child exited with signal number %d\n", WSTOPSIG(status));

        if ((second_childpid = fork()) == -1)
        {
            perror("Can`t fork.\n");
            return 1;
        }
        else if (second_childpid == 0)
        {
            printf("2nd child proc:\n");
            printf("\tPID: %d\n", getpid());
            printf("\tPPID: %d\n", getppid());
```

```

    printf("\tPGRP: %d\n", getpgrp());

    return 0;
}

second_childpid = wait(&status);
printf("2nd child has finished: PID = %d\n", second_childpid);

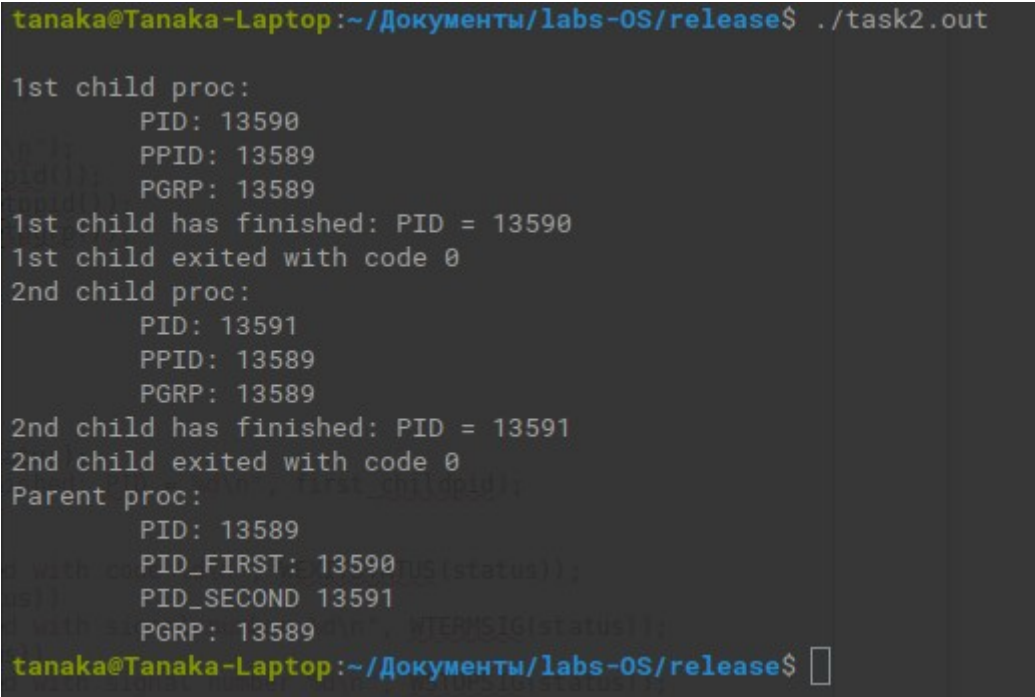
if (WIFEXITED(status))
    printf("2nd child exited with code %d\n", WEXITSTATUS(status));
else if (WIFSIGNALED(status))
    printf("2nd child exited with signal number %d\n", WTERMSIG(status));
else if (WIFSTOPPED(status))
    printf("2nd child exited with signal number %d\n", WSTOPSIG(status));
printf("Parent proc:\n");
printf("\tPID: %d\n", getpid());
printf("\tPID_FIRST: %d\n\tPID_SECOND %d\n", first_childpid, second_childpid);
printf("\tPGRP: %d\n", getpgrp());

return 0;
}
}

```

Пример работы программы

В данном задании была рассмотрена функция `wait()`. На рисунке 3 представлен пример работы программы.



```

tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$ ./task2.out

1st child proc:
    PID: 13590
    PPID: 13589
    PGRP: 13589
1st child has finished: PID = 13590
1st child exited with code 0
2nd child proc:
    PID: 13591
    PPID: 13589
    PGRP: 13589
2nd child has finished: PID = 13591
2nd child exited with code 0
Parent proc:
    PID: 13589
    PID_FIRST: 13590
    PID_SECOND 13591
    PGRP: 13589
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$ 

```

Изображение 3. – Работа второй программы.

В программе создаются два дочерних процесса, выводят информацию и завершаются до предка.

Собственный идентификатор (PID_FIRST = 13590) первого дочернего процесса, на единицу больше собственного идентификатора предка (PID = 13589), равного.

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.

Код программы

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    pid_t first_childpid, second_childpid;

    first_childpid = fork();

    if (first_childpid == -1)
    {
        perror("Can`t fork.\n");
        return 1;
    }
    else if (first_childpid == 0)
    {
        printf("\n1st child proc:\n");
        printf("\tPID: %d\n", getpid());
        printf("\tPPID: %d\n", getppid());
        printf("\tPGRP: %d\n", getpgrp());

        printf("Current PS:\n");
        execlp("ps", "", NULL);

        return 0;
    }
    else
    {
        int status;

        first_childpid = wait(&status);
        printf("1st child has finished: PID = %d\n", first_childpid);

        if (WIFEXITED(status))
            printf("1st child exited with code %d\n", WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
```

```

        printf("1st child exited with signal number %d\n", WTERMSIG(status));
    else if (WIFSTOPPED(status))
        printf("1st child exited with signal number %d\n", WSTOPSIG(status));

    if ((second_childpid = fork()) == -1)
    {
        perror("Can't fork.\n");
        return 1;
    }
    else if (second_childpid == 0)
    {
        printf("2nd child proc:\n");
        printf("\tPID: %d\n", getpid());
        printf("\tPPID: %d\n", getppid());
        printf("\tPGRP: %d\n", getpgrp());

        printf("OS:\n");
        execlp("neofetch", "", NULL);

        return 0;
    }

    second_childpid = wait(&status);
    printf("2nd child has finished: PID = %d\n", second_childpid);

    if (WIFEXITED(status))
        printf("2nd child exited with code %d\n", WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        printf("2nd child exited with signal number %d\n", WTERMSIG(status));
    else if (WIFSTOPPED(status))
        printf("2nd child exited with signal number %d\n", WSTOPSIG(status));

    printf("Parent proc:\n");
    printf("\tPID: %d\n", getpid());
    printf("\tPID_FIRST: %d\n\tPID_SECOND %d\n", first_childpid, second_childpid);
    printf("\tPGRP: %d\n", getpgrp());

    return 0;
}
}

```


Пример работы программы

В данном задании рассмотрена функция `exes()`. Пример работы программы представлен на рисунке 4.

```
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$ ./task3.out

1st child proc:
    PID: 14490
    PPID: 14489
    PGRP: 14489
Current PS:
    PID TTY          TIME CMD
    8738 pts/1        00:00:00 bash
    14489 pts/1        00:00:00 task3.out
    14490 pts/1        00:00:00 ps
1st child has finished: PID = 14490
1st child exited with code 0
2nd child proc:
    PID: 14491
    PPID: 14489
    PGRP: 14489

OS:
    OS: elementary OS 5.1.7 Hera x86_64
    Host: U38N 1.0
    Kernel: 5.4.0-58-generic
    Uptime: 1 hour, 23 mins
    Packages: 2464
    Shell: bash 4.4.20
    Resolution: 1920x1080
    DE: Pantheon
    WM: Mutter(Gala)
    Theme: Elementary [GTK3]
    Icons: Elementary [GTK3]
    Terminal: io.elementary.t
    CPU: AMD A8-4555M APU (4) @ 1.600GHz
    GPU: AMD Radeon HD 7600G
    Memory: 1410MiB / 3378MiB

=====
2nd child has finished: PID = 14491
2nd child exited with code 0
Parent proc:
    PID: 14489
    PID_FIRST: 14490
    PID_SECOND: 14491
    PGRP: 14489
```

Изображение 4. – Работа третьей программы.

В программе два дочерних процесса. В родительском процессе используется системный вызов wait(). В дочерних процессах используется системный вызов exes(). Была выведена информация PS и команды neofetch с информацией о компьютере.

Задание 4

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

Код программы

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define CNT 2
#define SIZE_RES 41
#define SIZE_MSG_F 17
#define SIZE_MSG_S 19

int main()
{
    int descr[CNT];

    if (pipe(descr) == -1)
    {
        printf("Can`t pipe\n");
        return 1;
    }

    char result_data[SIZE_RES];

    pid_t second_childpid;
    pid_t first_childpid = fork();

    if (first_childpid == -1)
    {
        perror("Can`t fork.\n");
        return 1;
    }
    else if (first_childpid == 0)
    {
        close(descr[0]);
        if (!write(descr[1], "Bark! from first\n", SIZE_MSG_F))
        {
            printf("Can`t write string\n");
            return 1;
        }
        return 0;
    }
    else
    {
        int status;
```

```

first_childpid = wait(&status);
printf("1st child has finished:\n\tPID = %d\n", first_childpid);

if (WIFEXITED(status))
    printf("1st child exited with code %d\n", WEXITSTATUS(status));
else if (WIFSIGNALED(status))
    printf("1st child exited with signal number %d\n", WTERMSIG(status));
else if (WIFSTOPPED(status))
    printf("1st child exited with signal number %d\n", WSTOPSIG(status));

if ((second_childpid = fork()) == -1)
{
    perror("Can't fork.\n");
    return 1;
}
else if (second_childpid == 0)
{
    close(descr[0]);
    if (!write(descr[1], "Bark! from second\n", SIZE_MSG_S))
    {
        printf("Can't write string\n");
        return 1;
    }
    return 0;
}

second_childpid = wait(&status);
printf("2nd child has finished:\n\tPID = %d\n", second_childpid);

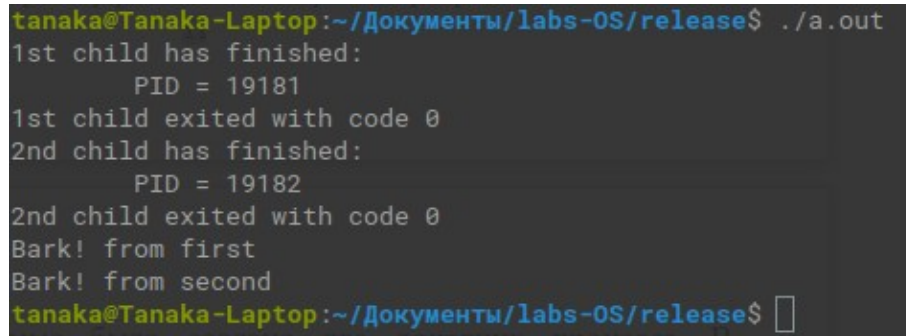
if (WIFEXITED(status))
    printf("2nd child exited with code %d\n", WEXITSTATUS(status));
else if (WIFSIGNALED(status))
    printf("2nd child exited with signal number %d\n", WTERMSIG(status));
else if (WIFSTOPPED(status))
    printf("2nd child exited with signal number %d\n", WSTOPSIG(status));

close(descr[1]);
if (read(descr[0], result_data, SIZE_RES) < 0)
{
    printf("Can't read string\n");
    return 1;
}
printf("%s", result_data);
return 0;
}
}

```

Пример работы программы

В данной программе был рассмотрен обмен сообщений между предком и потомком через программный канал. Пример работы программы представлен ниже.



```
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$ ./a.out
1st child has finished:
    PID = 19181
1st child exited with code 0
2nd child has finished:
    PID = 19182
2nd child exited with code 0
Bark! from first
Bark! from second
tanaka@Tanaka-Laptop:~/Документы/labs-OS/release$
```

Изображение 5. – Работа четвертой программы.

В данной программе было создано два дочерних процесса. В родительском процессе используется системный вызов `wait()`.

Потомки отправили предку сообщения.

Первый потомок – “Bark! from first”, второй потомок – “Bark! from second”.

Предок считывает данные их буфера и выводит на консоль.

Задание 5

В программу с программным каналом включить собственный обработчик сигнала.

Использовать сигнал для изменения хода выполнения программы.

Код программы

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>

#define SIZE_RES 36
#define SIZE_MSG_F 17
#define SIZE_MSG_S 19

typedef int flag_s;

flag_s sigflag = 0;
void sigcatcher(int signum)
{
    printf("\nSignal number %d catched!\n", signum);
    sigflag = 1;
}

// descr он же fd - имеет два дескриптора 0 - для чтения, 1 - для записи
int main()
{
    // устанавливаем реакцию на сигнал, вызываемую функцию
    signal(SIGTSTP, sigcatcher);
    // массив файловых дескрипторов
    int descr[2];

    if (pipe(descr) == -1)
    {
        printf("Can't pipe\n"); // Не удалось
        return 1;
    }

    // массива результата
    char result_data[SIZE_RES];
    pid_t second_childpid;
    pid_t first_childpid = fork(); // форкаем

    if (first_childpid == -1)
    {
        perror("Can't fork.\n");
        return 1; // Не вышло
    }
    else if (first_childpid == 0)
    {
        // Находимся в первом потомке
```

```

if (!sigflag) // Если ни разу сигнала не было, то выводим
{
    printf("You have 5 second to use Ctrl+Z\n");
    sleep(5);
}
// Если все же за этот момент времени сигнал поступил
if (sigflag)
{
    //Пишем в канал
    close(descr[0]); // Из канала читать нельзя, если в него пишут
    // в канал писать нельзя, если его читают
    if (!write(descr[1], "Bark! from first\n", SIZE_MSG_F))
    {
        printf("Can't write string\n");
        return 1;
    }
    else
        printf("String writed from first!\n");
}
return 0;
}
else
{
    int status;
    first_childpid = wait(&status);
    printf("1st child has finished:\n\tPID = %d\n", first_childpid);

    if (WIFEXITED(status))
        printf("1st child exited with code %d\n", WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        printf("1st child exited with signal number %d\n", WTERMSIG(status));
    else if (WIFSTOPPED(status))
        printf("1st child exited with signal number %d\n", WSTOPSIG(status));

    if ((second_childpid = fork()) == -1)
    {
        perror("Can't fork.\n");
        return 1;
    }
    else if (second_childpid == 0)
    {
        if (sigflag)
        {
            close(descr[0]);
            if (!write(descr[1], "Bark! from second\n", SIZE_MSG_S))
            {
                printf("Can't write string\n");
                return 1;
            }
            else
                printf("String writed from second!\n");
        }
        return 0;
    }
}

second_childpid = wait(&status);
printf("2nd child has finished:\n\tPID = %d\n", second_childpid);

if (WIFEXITED(status))
    printf("2nd child exited with code %d\n", WEXITSTATUS(status));
else if (WIFSIGNALED(status))
    printf("2nd child exited with signal number %d\n", WTERMSIG(status));
else if (WIFSTOPPED(status))
    printf("2nd child exited with signal number %d\n", WSTOPSIG(status));

if (sigflag)
{

```

```

        close(descr[1]);
        if (read(descr[0], result_data, SIZE_RES) < 0)
        {
            printf("Can`t read string\n");
            return 1;
        }
    }
    else
    {
        strcpy(result_data, "Time is out!\n\n");
    }

    printf("%s", result_data);

    return 0;
}
}

```

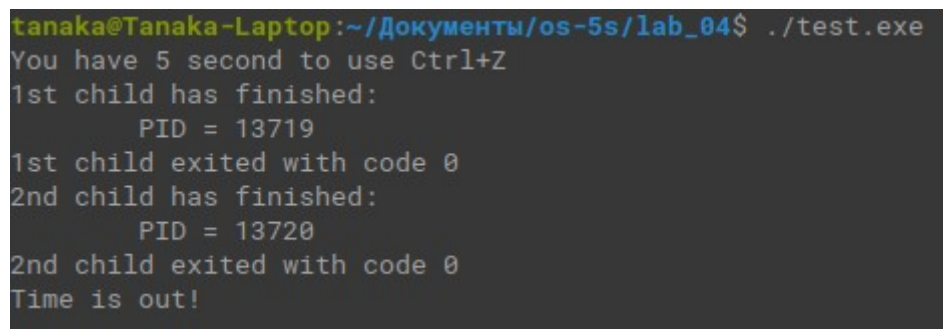
Пример работы программы

В данной программе добавлен обработчик сигнала прерывания (Ctrl-Z) с терминала.

В данной программе два дочерних процесса.

В родительском процессе используется системный вызов wait(). Предок и потомки обмениваются сообщением через программный канал.

Если в течение 5 секунд нажать комбинацию клавиш Ctrl Z, тогда по сигналу будут записаны в буфер строки сообщений потомков, которые предок сохранит в result, иначе отсчет будет продолжаться, пока не появится сообщение «Time is out!».



```

tanaka@Tanaka-Laptop:~/Документы/os-5s/lab_04$ ./test.exe
You have 5 second to use Ctrl+Z
1st child has finished:
    PID = 13719
1st child exited with code 0
2nd child has finished:
    PID = 13720
2nd child exited with code 0
Time is out!

```

Изображение 6. – Работа программы, если не было сигнала.

```
tanaka@Tanaka-Laptop:~/Документы/os-5s/lab_04$ ./test.exe
You have 5 second to use Ctrl+Z
^Z
Signal number 20 cathed!
String writed from first!

Signal number 20 cathed!
1st child has finished:
    PID = 13816
1st child exited with code 0
String writed from second!
2nd child has finished:
    PID = 13817
2nd child exited with code 0
Bark! from first
Bark! from second
```

Изображение 7. – Работа программы, если сигнал все же был.