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***		**Чыонг Н. В. У.**
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			**Рязанова Н. Ю.**
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021г

❖ **Задание 1:**

Написать программу, запускающую не мене 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.

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

#define DELAY 4

int main() {
    pid_t child1 = fork();

    if (child1 == -1)
    {
        perror("Can't fork Child 1");
        exit(1);
    }
    else if (child1 == 0) // Child 1
    {
        printf("Child 1: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
        sleep(DELAY);
        printf("Child 1: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
        return 0;
    }

    pid_t child2 = fork();
    if (child2 == -1)
    {
        perror("Can't fork");
        exit(1);
    }
    else if (child2 == 0) // Child 2
    {
        printf("Child 2: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
        sleep(DELAY);
        printf("Child 2: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
        return 0;
    }

    // Parent
    printf("Parent: pid=%d, child1=%d, child2=%d, groupid=%d\n", getpid(), child1, child2, getpgrp());
    return 0;
}
```

```
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.

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

#define DELAY 5

int main() {
    // Fork Child 1
    pid_t child1 = fork();
    if (child1 == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child1 == 0)
    {
        printf("Child 1: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp() );
        sleep(DELAY);
        printf("Child 1: Exiting\n");
        return 0;
    }

    // Fork Child 2
    pid_t child2 = fork();
    if (child2 == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child2 == 0)
    {
        printf("Child 2: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
        sleep(DELAY);
        printf("Child 2: Exiting\n");
        return 0;
    }

    // Parent
    int status1;
    pid_t ret1 = wait(&status1);
    int status2;
    pid_t ret2 = wait(&status2);

    printf("Parent: pid = %d, group = %d, child1 = %d, Child2 = %d\n", getpid(), getpgrp(), child1, child2 );

    if (WIFEXITED(status1))
        printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
    else if (WIFSIGNALED(status1))
        printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
    else if (WIFSTOPPED(status1))
        printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));

    if (WIFEXITED(status2))
        printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
    else if (WIFSIGNALED(status2))
        printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
    else if (WIFSTOPPED(status2))
        printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));

    return 0;
}
```

```
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.

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

int main() {
    // Child 1
    pid_t child1 = fork();
    if (child1 == -1)
    {
        perror("Child 1: Couldn't fork.");
        exit(1);
    }
    else if (child1 == 0)
    {
        printf("Child 1: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());

        if (execlp("ps", "ps", "al", 0) == -1)
        {
            perror("Child 1: couldn't exec.");
            exit(1);
        }
    }
    // Child 2
    pid_t child2 = fork();
    if (child2 == -1)
    {
        perror("Child 2: Couldn't fork.");
        exit(1);
    }
    else if (child2 == 0)
    {
        printf("Child 2: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());

        if (execlp("ls", "ls", "-ail", 0) == -1)
        {
            perror("Child 2: couldn't exec.");
            exit(1);
        }
    }
    // Parent
    if (child1 != 0 && child2 != 0)
    {
        int status1, status2;
        pid_t ret1 = wait(&status1);
        pid_t ret2 = wait(&status2);

        printf("Parent: pid=%d, groupid=%d, child1=%d, child2=%d\n", getpid(), getpgrp(), child1, child
2);

        if (WIFEXITED(status1))
            printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
```

```c
        else if (WIFSIGNALED(status1))
            printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
        else if (WIFSTOPPED(status1))
            printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));

        if (WIFEXITED(status2))
            printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
        else if (WIFSIGNALED(status2))
            printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
        else if (WIFSTOPPED(status2))
            printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
    }

    return 0;
}
```

```
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_3$ vim task_3.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/lab_04/task_3$ gcc -std=c99 -o task_3.exe task_3.c
tnvu4920@tnvu4920-VirtualBox:~/iu7-52b/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 -rwxrwxr-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_pol Sl+ tty2       0:59 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
0  1000    1254    1192  20   0 199304 15060 poll_s Sl+  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  20052  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:~/iu7-52b/lab_04/task_3$
```

❖ **Задание 4:**

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

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

#define LEN 100

int main() {
    // Pipe
    int fd[2];
    if (pipe(fd) == -1)
    {
        perror("Couldn't pipe.");
        exit(1);
    }
    // Child 1
    pid_t child1 = fork();
    if (child1 == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child1 == 0)
    {
        close(fd[0]);
        char msg1[] = "Hello From Child 1 ";
        write(fd[1], msg1, LEN);
        printf("Child 1: writen [%s] to Pipe\n", msg1);
        exit(0);
    }
    // Child 2
    int child2 = fork();
    if (child2 == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child2 == 0)
    {
        close(fd[0]);
        char msg2[] = "Hello From Child 2 ";
        write(fd[1], msg2, LEN);
        printf("Child 2: writen [%s] to Pipe\n", msg2);
        exit(0);
    }
    // Parentss

    int status1, status2;
    pid_t ret1 = wait(&status1);
    pid_t ret2 = wait(&status2);

    close(fd[1]);
    char msg1[LEN];
    read(fd[0], msg1, LEN);
    char msg2[LEN];
    read(fd[0], msg2, LEN);
    printf("Parent: read from Pipe [%s%s]\n", msg1, msg2);
    if (WIFEXITED(status1))
        printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
    else if (WIFSIGNALED(status1))
        printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
    else if (WIFSTOPPED(status1))
        printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));

```

```c
    if (WIFEXITED(status2))
        printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
    else if (WIFSIGNALED(status2))
        printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
    else if (WIFSTOPPED(status2))
        printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
    return 0;
}
```

```
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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <sys/wait.h>

#define LEN 100
#define DELAY 2

int p_flag = 0;

void catch_sig(int signum) {
    printf("Proccess %d: catched signal %d\n", getpid(), signum);
    p_flag = 1;
}

int main() {
    //Pipe
    int fd[2];
    if (pipe(fd) == -1)
    {
        perror("Couldn't pipe.");
        exit(1);
    }
    // Ctrl + Z
    void (*old_handler)(int) = signal(SIGTSTP, catch_sig);

    // Child 1
    pid_t child1 = fork();
    if (child1 == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child1 == 0)
    {
        while (!p_flag) ;
        close(fd[0]);
        char msg1[] = "Hello From Child 1 ";
        write(fd[1], msg1, LEN);
        printf("Child 1: writen [%s] to Pipe\n", msg1);
    }

    // Child 2
    pid_t child2 = fork();
```

```c
    if (child2 == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child2 == 0)
    {
        while (!p_flag) ;
        close(fd[0]);
        char msg2[] = "Hello From Child 2 ";
        write(fd[1], msg2, LEN);
        printf("Child 2: writen [%s] to Pipe\n", msg2);
    }
    // Parent
    if (child1 != 0 && child2 != 0)
    {
        printf("Parent: pid = %d\n", getpid());
        printf("Child 1: pid = %d\n", child1);
        printf("Child 2: pid = %d\n\n", child2);
        printf("Parent: waiting for CTRL+Z signal\n");
        if  e (!p_flag);

        int status1, status2;
        pid_t ret1 = wait(&status1);
        pid_t ret2 = wait(&status2);

        printf("------------\n");
        close(fd[1]);
        char msg1[LEN], msg2[LEN];
        read(fd[0], msg1, LEN);
        read(fd[0], msg2, LEN);
        printf("Parent: read from Pipe [%s%s]\n", msg1, msg2);

        if (WIFEXITED(status1))
            printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
        else if (WIFSIGNALED(status1))
            printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
        else if (WIFSTOPPED(status1))
            printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));

        if (WIFEXITED(status2))
            printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
        else if (WIFSIGNALED(status2))
            printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
        else if (WIFSTOPPED(status2))
            printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));

    }
    signal(SIGTSTP, old_handler);
    return 0;
}
```

```
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
^ZProccess 9372: catched signal 20
Proccess 9374: catched signal 20
Child 2: writen [Hello From Child 2 ] to Pipe
Proccess 9373: catched 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$
```