**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-56Б*** | | **Нгуен Ф. С.** |
| | (Группа) | (Подпись, дата) | (И.О. Фамилия) |
| Преподаватель | | | **Рязанова Н. Ю.** |
| | | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2020*

❖ **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;
}
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$ ./1.exe
Parent: pid=18408, child1=18409, child2=18410, groupid=18408
Child 1: pid=18409, ppid=18408, groupid=18408
Child 2: pid=18410, ppid=18408, groupid=18408
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$ Child 1: pid=18409, ppid=1457, groupid=18408
Child 2: pid=18410, ppid=1457, groupid=18408
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$
```

❖ **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;
}
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$ ./2.exe
Child 1: pid = 18878, ppid = 18877, groupid = 18877
Child 2: pid = 18879, ppid = 18877, groupid = 18877
Child 1: Exiting
Child 2: Exiting
Parent: pid = 18877, group = 18877, child1 = 18878, Child2 = 18879
Parent: child 18878 finished with 0 code.
Parent: child 18879 finished with 0 code.
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$
```

### ❖ 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);

```

```c
        printf("Parent: pid=%d, groupid=%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;
}
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$ ./3.exe
Child 1: pid = 20053, ppid = 20052, groupid = 20052
Child 2: pid = 20054, ppid = 20052, groupid = 20052
total 112
2228461 drwxrwxr-x 2 nguyensang nguyensang  4096 дек  2 12:32 .
2228248 drwxrwxr-x 4 nguyensang nguyensang  4096 ноя 30 16:10 ..
2228357 -rwxrw-rw- 1 nguyensang nguyensang   951 дек  2 11:50 1.c
2228328 -rwxrwxr-x 1 nguyensang nguyensang 17000 дек  2 11:50 1.exe
2228901 -rwxrw-rw- 1 nguyensang nguyensang  1681 дек  2 12:11 2.c
2228898 -rwxrwxr-x 1 nguyensang nguyensang 17136 дек  2 12:03 2.exe
2228643 -rwxrw-rw- 1 nguyensang nguyensang  1730 дек  2 12:32 3.c
2228289 -rwxrwxr-x 1 nguyensang nguyensang 17096 дек  2 12:32 3.exe
2228648 -rwxrw-rw- 1 nguyensang nguyensang  1902 дек  2 01:34 4.c
2228905 -rwxrwxr-x 1 nguyensang nguyensang 17088 дек  2 01:01 4.exe
2228968 -rwxrw-rw- 1 nguyensang nguyensang  1061 мая 28  2020 5.c
2228907 -rw-rw-r-- 1 nguyensang nguyensang   192 дек  2 12:25 makefile
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4  1000    1507    1449  20   0 164016  6304 poll_s Ssl+ tty2       0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE
4  1000    1514    1507  20   0 287652 68528 ep_pol Sl+  tty2       1:59 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthori
0  1000    1569    1507  20   0 190988 15560 poll_s Sl+  tty2       0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubunt
0  1000    4505    4488  20   0  10732  4996 do_wai Ss   pts/1      0:00 bash
0  1000   20052    4505  20   0   2356   524 do_wai S+   pts/1      0:00 ./3.exe
0  1000   20053   20052  20   0  11416  3212 -      R+   pts/1      0:00 ps al
Parent: pid=20052, groupid=20052, child1=20053, child2=20054
Parent: child 20054 finished with 0 code.
Parent: child 20053 finished with 0 code.
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$
```

❖ **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));

```

```
    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;
}
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$ ./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 20665 finished with 0 code.
Parent: child 20666 finished with 0 code.
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$
```

**Задание 5:**

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

```
#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);

    pid_t child = fork();
    if (child == -1)
    {
        perror("Couldn't fork.");
        exit(1);
    }
    else if (child == 0) // Child
    {
        while (!p_flag) ;
        char msg[LEN];
        close(fd[1]);
        read(fd[0], msg, LEN);
        printf("Child: read [%s]\n", msg);
        sleep(DELAY);
    }
```

```
45.     else // Parent
46.     {
47.         printf("Parent: pid = %d\n", getpid());
48.         printf("Child: pid = %d\n", child);
49.         close(fd[0]);
50.         char msg[] = "Hello From Parent ";
51.         write(fd[1], msg, LEN);
52.
53.         printf("Parent: waiting for CTRL+Z signal\n");
54.         while (!p_flag);
55.
56.         int status;
57.         pid_t ret = wait(&status);
58.
59.         if (WIFEXITED(status))
60.             printf("Parent: child %d finished with %d code.\n", ret, WEXITSTATUS(status));
61.         else if (WIFSIGNALED(status))
62.             printf("Parent: child %d finished from signal with %d code.\n", ret, WTERMSIG(status));
63.         else if (WIFSTOPPED(status))
64.             printf("Parent: child %d finished from signal with %d code.\n", ret, WSTOPSIG(status));
65.     }
66.
67.     signal(SIGTSTP, old_handler);
68.     return 0;
69. }
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$ ./5.exe
Parent: pid = 23989
Child: pid = 23990
Parent: waiting for CTRL+Z signal
^ZProccess 23989: catched signal 20
Proccess 23990: catched signal 20
Child: read [Hello From Parent ]
Parent: child 23990 finished with 0 code.
nguyensang@K-virtual-machine:~/Desktop/OS2020/2$
```