|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет**

***к лабораторной работе №4***

***по дисциплине «Операционные системы»***

***по теме «Процессы. Системные вызовы.»***

|  |  |
| --- | --- |
| **Студент: Русинова Дарья**  **Группа: ИУ7-55б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Рязанова Н. Ю.** |  |

Москва.

2020 г.

**Задание 1.**

Написать программу, запускающую не мене двух новых процессов системным вызовом fork(). В предке вывести собственный идентификатор, идентификатор группы и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка и идентификатор группы.

Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// в программе создается три процесса-потомка от одного процесса-предка

int main()

{

    printf("3 children are created in following program.\n\n");

    // пусть будет создано три потомка

    for (int i = 0; i < 3; i++)

    {

        int child = fork();

        if (child == -1)

        {

            printf("Can't fork child\n");

            return 1;

        }

        if (child == 0)

        {

            // код процесса-потомка

            sleep(1);

            printf("Child: pid=%d, parentpid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());

            // чтобы предок завершился раньше своих потомков

            sleep(10);

            return 0;

        }

        else

        {

            // код процесса-предка

            sleep(2);

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

            // завершаем процесс предок после последней итерации

            if (i == 2)

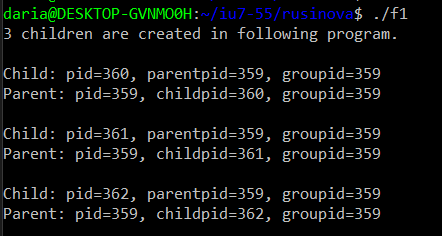
                return 0;

        }

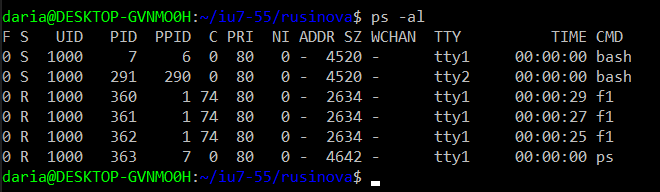
    }

}

Вывод программы.



Процесс-предок завершился раньше процессов-потомков. Потомки получили идентификатор PPID, равный 1.



**Задание 2.**

Написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов wait().

// в программе создается три процесса-потомка от одного процесса-предка

int main()

{

    printf("3 children are created in following program.\n\n");

    // будет создано три потомка

    for (int i = 0; i < 3; i++)

    {

        int child = fork();

        if (child == -1)

        {

            printf("Can't fork child\n");

            return 1;

        }

        if (child == 0)

        {

            // код процесса-потомка

            sleep(1);

            printf("Child: pid=%d, pidid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());

            return 0;

        }

        else

        {

            // код процесса-предка

            // ожидаем завершения процесса-потомка

            int status;

            int ch\_pid = wait(&status);

            // интерпретация значения status

            // дочерний процесс завершен нормально

            if (WIFEXITED(status))

                printf("Child with pid = %d has finished with exit code %d\n", ch\_pid, WEXITSTATUS(status));

            // дочерний процесс завершился неперехватываемым сигналом

            else if (WIFSIGNALED(status))

                printf("Child with pid = %d has finished by signal %d\n", ch\_pid, WTERMSIG(status));

            // дочерний процесс остановился

            else if (WIFSTOPPED(status))

                printf("Child with pid = %d has been stopped by signal %d\n", ch\_pid, WSTOPSIG(status));

            sleep(2);

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

            // завершаем процесс-предок на последней итерации

            if (i == 2)

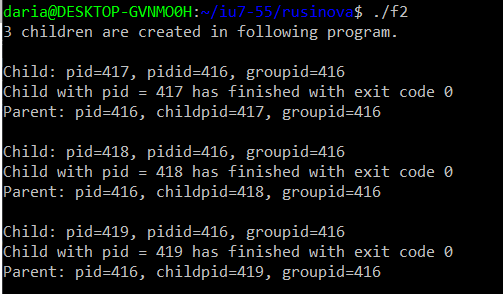
                return 0;

        }

    }

}

Вывод программы:



**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.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h> //wait

// в программе создается два процесса-потомка от одного процесса-предка

// первый процесс-потомок выполняет ps, второй процесс потомок выполняет ls

int main()

{

    printf("2 children are created in following program.\n\n");

    int children[2] = {-1, -1};

    char\* exec\_paths[2] = {"./usr/bin/ps", "./usr/bin/ls"};

    char\* exec\_calls[2] = {"ps", "ls"};

    char\* exec\_params[2] = {"-al", "-l"};

    // будет создано два потомка

    for (int i = 0; i < 2; i++)

    {

        children[i] = fork();

        if (children[i] == -1)

        {

            printf("Can't fork child\n");

            return 1;

        }

        if (children[i] == 0)

        {

            // код процесса-потомка

            sleep(1);

            printf("Child: pid=%d, pidid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());

            // процесс-потомок вызывает системный вызов execl

            if (execl(exec\_paths[i], exec\_calls[i], exec\_params[i], (char\*)NULL) == -1)

                printf("Child with pid=%d failed to exec\n", getpid());

            return 0;

        }

        else

        {

            // код процесса-предка

            // ожидаем завершения процесса-потомка

            int status;

            int ch\_pid = wait(&status);

            // интерпретация значения status

            // дочерний процесс завершен нормально

            if (WIFEXITED(status))

                printf("Child with pid = %d has finished with exit code %d\n", ch\_pid, WEXITSTATUS(status));

            // дочерний процесс завершился неперехватываемым сигналом

            else if (WIFSIGNALED(status))

                printf("Child with pid = %d has finished by signal %d\n", ch\_pid, WTERMSIG(status));

            // дочерний процесс остановился

            else if (WIFSTOPPED(status))

                printf("Child with pid = %d has been stopped by signal %d\n", ch\_pid, WSTOPSIG(status));

            sleep(2);

            printf("Parent: pid=%d, childpid=%d, groupid=%d\n\n", getpid(), children[i], getpgrp());

            // завершаем процесс-предок на последней итерации

            if (i == 2)

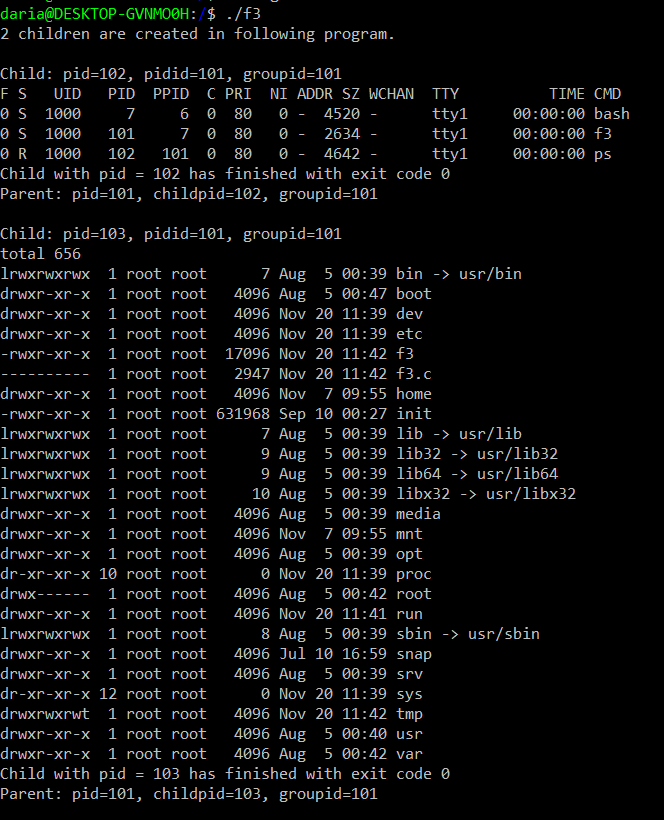
                return 0;

        }

    }

}

Вывод программы:



Задание 4.

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

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h> //wait

int main()

{

    // заводим массив файловых дескрипторов

    // массив файловыйх дескрипторов - средство взаимоисключения

    // необходимо для организации монопольного доступа:

    // в канал нельзя писать, если из него читают;

    // нельзя читать, если в него пишут

    // descr[0] - на чтение

    // descr[1] - на запись

    // Канал уничтожается, когда закрыты все файловые дескрипторы,

    // ссылающиеся на него.

    // массив из двух файловых дескрипторов

    int descr[2];

    if (pipe(descr) == -1)

    {

        printf("Failed to pipe.\n");

        return 1;

    }

    printf("2 children are created in following program.\n\n");

    int children[2] = {-1, -1};

    // будет создано два потомка

    for (int i = 0; i < 2; i++)

    {

        // создание потомка

        children[i] = fork();

        if (children[i] == -1)

        {

            printf("Can't fork child\n");

            return 1;

        }

        if (children[i] == 0)

        {

            // код процесса-потомка

            // подготовка сообщения

            char msg[50];

            sprintf(msg, "Message from child with pid = %d", getpid());

            // процесс-потомок передает сообщение

// процессу предку через программный канал

            close(descr[0]); // закрываем на чтение

            if (write(descr[1], msg, 50) > 0)

                printf("Child with pid = %d, ppid = %d has written to pipe\n",

getpid(), getppid()); // пишем в канал

            return 0;

        }

    }

    if (children[0] > 0 && children[1] > 0)

    {

        // код процесса - предка

        // ожидание завершения обоих процессов-потомков

        int st1, st2;

        if (waitpid(children[0], &st1, 0) == -1)

        {

            printf("Error while waiting for child with pid = %d", children[0]);

            return 1;

        }

        if (waitpid(children[1], &st2, 0) == -1)

        {

            printf("Error while waiting for child with pid = %d", children[1]);

            return 1;

        }

        // закрытие программного канала на запись

        close(descr[1]);

        char msg1[50];

        char msg2[50];

        // чтение сообщений от обоих потомков

        read(descr[0], msg1, 50);

        read(descr[0], msg2, 50);

        // выводы сообщений от потомков в консоль

        printf("Parent with pid = %d has read from pipe: %s\n", getpid(), msg1);

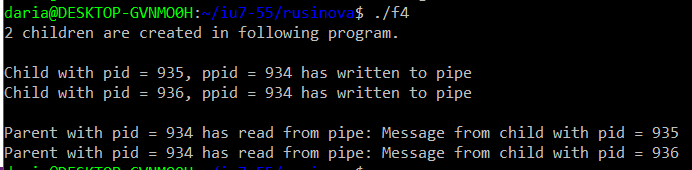
        printf("Parent with pid = %d has read from pipe: %s\n", getpid(), msg2);

        return 0;

    }

}

Вывод программы.



**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 <sys/types.h>

#include <sys/wait.h> //wait

// в данной программе написан обработчик сигнала SIGTSTP

// (Ctrl + Z с клавиатуры)

// если сигнал был получен, процессы-потомки передают процессу-предку

// соответсвующие сообщения, процесс-предок выводит сообщения на экран

// если сигнал не был получен, сообщение не передается

int is\_catched = 0;

void catcher\_SIGTSTP(int signum)

{

    printf("\n");

    is\_catched = 1;

}

int main()

{

    int descr[2];

    if (pipe(descr) == -1)

    {

        printf("Failed to pipe.\n");

        return 1;

    }

    // сохранение старого обработчика для дальнейшего восстановления

    void (\*old\_handler)(int) = signal(SIGTSTP, catcher\_SIGTSTP);

    // будет создано два потомка

    int children[2];

    for (int i = 0; i < 2; i++)

    {

        // создание потомка

        children[i] = fork();

        if (children[i] == -1)

        {

            printf("Can't fork child\n");

            return 1;

        }

        if (children[i] == 0)

        {

            // код процесса-потомка

            sleep(5);

            // если сигнал был получен

            if (is\_catched)

            {

                // подготовка сообщения

                char child\_msg[64];

                sprintf(child\_msg, "Message from child with pid = %d - signal was catched", getpid());

                // процесс-потомок передает сообщение

// процессу предку через программный канал

                close(descr[0]); // закрываем на чтение

                if (write(descr[1], child\_msg, 64) > 0)

                    printf("Child with pid = %d, ppid = %d has written to pipe\n", getpid(), getppid()); // пишем в канал

            }

            // если сигнал не был получен ничего не происходит

            return 0;

        }

    }

    if (children[0] > 0 && children[1] > 0)

    {

        // код процесса - предка

        // ожидание завершения обоих процессов-потомков

        int st1, st2;

        if (waitpid(children[0], &st1, 0) == -1)

        {

            printf("Error while waiting for child with pid = %d", children[0]);

            return 1;

        }

        if (waitpid(children[1], &st2, 0) == -1)

        {

            printf("Error while waiting for child with pid = %d", children[1]);

            return 1;

        }

        if (is\_catched)

        {

            // закрытие программного канала на запись

            close(descr[1]);

            char msg1[64];

            char msg2[64];

            // чтение сообщений от обоих потомков

            read(descr[0], msg1, 64);

            read(descr[0], msg2, 64);

            // выводы сообщений от потомков в консоль

            printf("Parent with pid = %d has read from pipe: %s",getpid(),msg1);

            printf("Parent with pid = %d has read from pipe: %s",getpid(),msg2);

        }

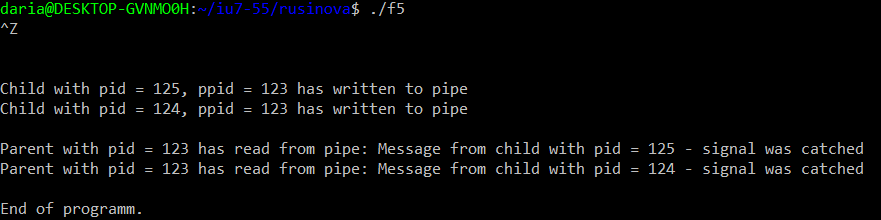
        printf("\nEnd of programm.\n");

        return 0;

    }

}

Вывод программы, если сигнал был получен: вывод сообщения о получении сигнала на экран.



Вывод программы, если сигнал не был получен: нет вывода сообщения.

