|  |  |
| --- | --- |
| **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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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());

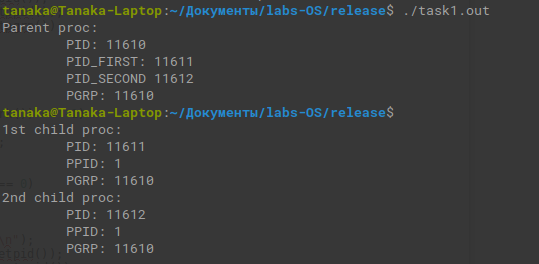
return 0;

}

}

**Пример работы программы**

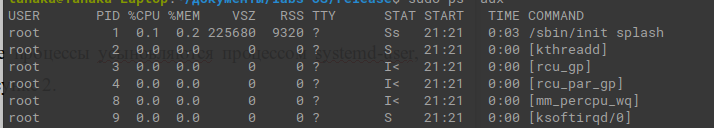
В данном задании было рассмотрено усыновление процессов.На изображении 1 представлен пример работы программы.

****

*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.

**

*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);

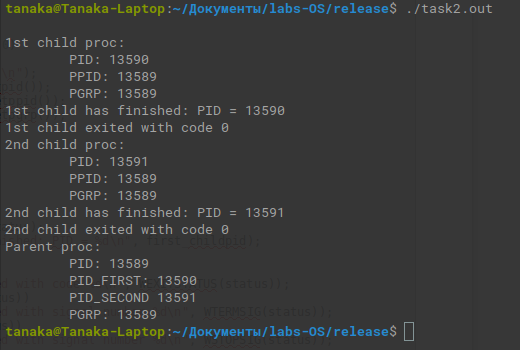
printf("\tPGRP: %d\n", getpgrp());

return 0;

}

}

**Пример работы программы**

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

*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());

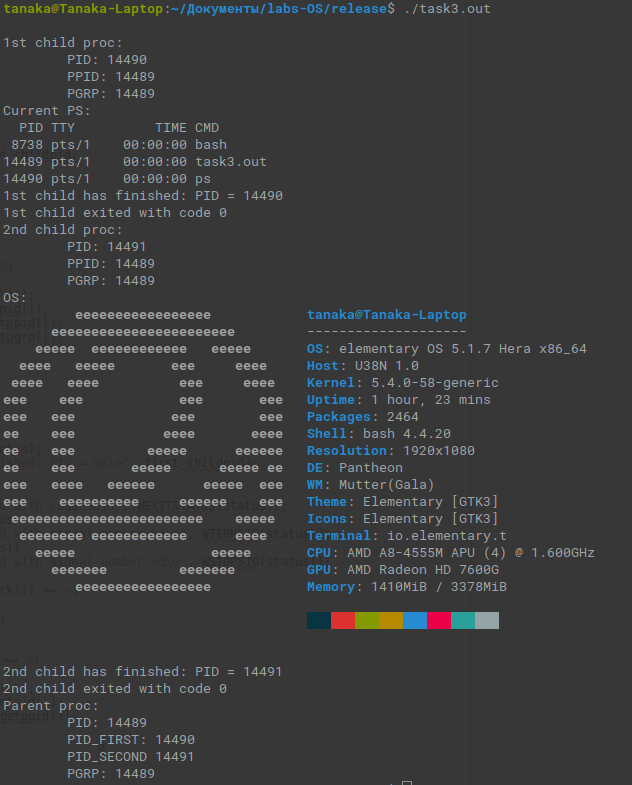
return 0;

}

}

**Пример работы программы**

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

****

*Изображение 4. – Работа третьей программы.*

В программе два дочерних процесса. В родительском процессе используется системный вызов wait(). В дочерних процессах используется системный вызов exec(). 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);

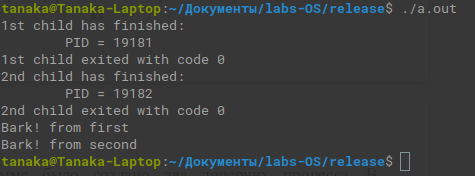
return 0;

}

}

**Пример работы программы**

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

**

*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 cathed!\n", signum);

sigflag = 1;

}

// descr он же fd - имеет два дескрипора 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 пистать 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;

}

}

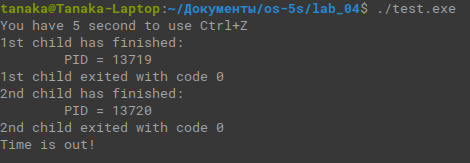
**Пример работы программы**

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

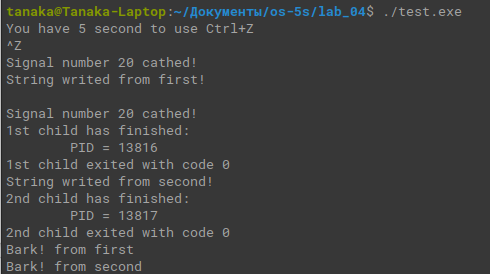
В данной программе два дочерних процесса.

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!».



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



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