Министерство образования Республики Беларусь Учреждение образования

«Брестский государственный технический университет» Кафедра ИИТ

Лабораторная работа №4

по дисциплине «Операционные системы»

Тема: «GCC. Процессы»

Вариант 2

Выполнил: 220203

студент 2-го курса

группы ПО-11

Антонюк Н.А.

Проверила:

Давидюк Ю.И.

Брест 2023

**Цель работы:** ознакомиться с компилятором GCC; изучить создание, завершение и изменение пользовательского контекста процессов в операционной системе UNIX.

**Ход работы**

***Задание для выполнения:***

Написать программу, которая будет реализовывать следующие функции: · сразу после запуска получает и сообщает свой ID и ID родительского процесса;

· перед каждым выводом сообщения об ID процесса и родительского процесса эта информация получается заново;

· порождает процессы, формируя генеалогическое дерево согласно варианту, сообщая, что «процесс с ID таким-то породил процесс с таким-то ID»; · перед завершением процесса сообщить, что «процесс с таким-то ID и таким то ID родителя завершает работу»;

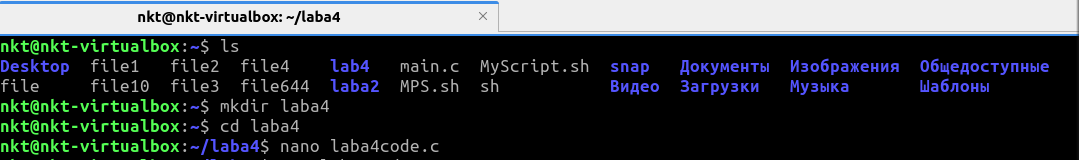
· один из процессов должен вместо себя запустить программу, указанную в варианте задания.

На основании выходной информации программы предыдущего пункта изобразить генеалогическое дерево процессов (с указанием идентификаторов процессов). Объяснить каждое выведенное сообщение и их порядок в предыдущем пункте.

***Вариант индивидуального задания:***

|  |  |  |  |
| --- | --- | --- | --- |
| *№* | *fork* | *exec* | |
| 2 | 0 1 2 2 3 4 6 | 2 | ps -a |

**Работа в консоли:**



**Код программы:**

#include <unistd.h>

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

int main()

{

pid\_t pid;

printf("Процесс 0 PID = %d PPID = %d\n", getpid(), getppid());

// Порождение первого процесса

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

// Порождение процесса 1

printf("Порождение процесса 1 PID = %d PPID = %d\n", getpid(), getppid());

// Порождение второго процесса

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

// Порождение процесса 2

printf("Порождение процесса 2 PID = %d PPID = %d\n", getpid(), getppid());

// Порождение процесса 3

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

printf("Порождение процесса 3 PID = %d PPID = %d\n", getpid(), getppid());

// Порождение процесса 5

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

printf("Порождение процесса 5 PID = %d PPID = %d\n", getpid(), getppid());

printf("Завершился процесс 5 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}

else

sleep(1);

printf("Завершился процесс 3 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}else sleep(2);

//stop 3

// Порождение процесса 4

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

printf("Порождение процесса 4 PID = %d PPID = %d\n", getpid(), getppid());

// Порождение процесса 6

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

printf("Порождение процесса 6 PID = %d PPID = %d\n", getpid(), getppid());

// Порождение процесса 7

if ((pid = fork()) == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

printf("Порождение процесса 7 PID = %d PPID = %d\n", getpid(), getppid());

printf("Завершился процесс 7 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}

else

sleep(1);

printf("Завершился процесс 6 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}

else sleep(2);

//stop 6

printf("Завершился процесс 4 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}else sleep(3);

printf("Завершился процесс 2 PID = %d PPID = %d\n", getpid(), getppid());

execl("/bin/ps", "ps", "-a", NULL);

perror("execl"); // Если execl завершилась с ошибкой

// Дальше код не выполнится, если execl выполнится успешно

exit(EXIT\_FAILURE);

}

else

sleep(7);

//stop 4

/\*printf("Завершился процесс 2 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}\*/

/\*}

else sleep(7);

printf("Завершился процесс 1 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);\*/

printf("Завершился процесс 1 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

}else sleep(9);

printf("Завершился процесс 0 PID = %d PPID = %d\n", getpid(), getppid());

exit(0);

return 0;

}

/\*int main(){

pid\_t pid;

printf("Процесс 0 PID = %d PPID = %d\n", getpid(), getppid());

if((pid = fork()) == 0)

printf("Порождение процесса 1 PID = %d PPID = %d\n", getpid(), getppid());

printf("Завершился процесс 1 PID = %d PPID = %d\n", getpid(), getppid());

}

else sleep(1);

printf("Завершился процесс 0 PID = %d PPID = %d\n", getpid(), getppid());

sleep(4);

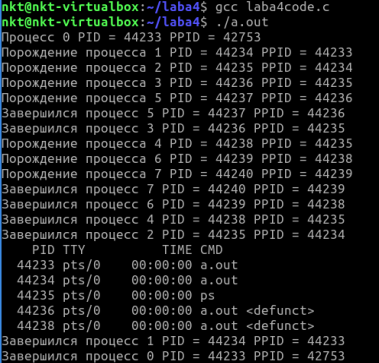
return 0;

}\*/

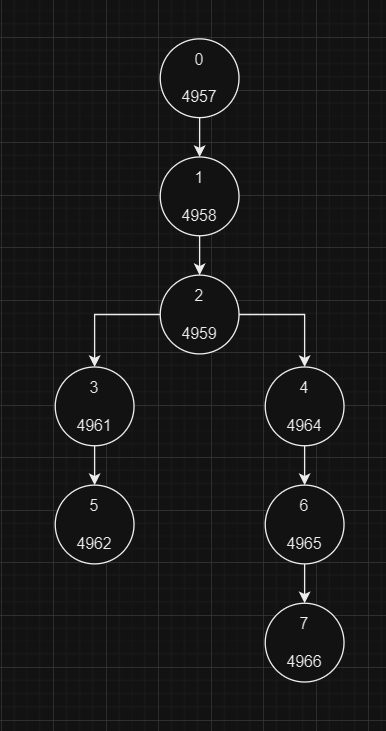
**Результат выполнения**

**программы:** gcc 1.c

./1.out



**Генеалогическое дерево процессов:**



**Вывод:** в ходе выполнения данной лабораторной работы были изучены основы создания, завершения и изменения пользовательского контекста процессов в операционной системе UNIX.