**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

отчет

**по лабораторной работе №5**

**по дисциплине «Операционные системы»**

**по теме: «Создание и уничтожение процессов»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6361 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Иванов А.И. |
| Преподаватель |  | Широков В.В. |

Санкт-Петербург

2019

Цель работы - знакомство с основными системными вызовами, обеспечивающими создание процессов.

Общие сведения

Основным системным вызовом для создания нового процесса в операционных системах, поддерживающих стандарт POSIX, является следующий вызов:

pid\_t fork(void).

Вызов fork(), сделанный в некотором процессе, который будем называть родительским, создает дочерний процесс, который является практически полной копией родительского процесса. При создании данные родительского процесса копируются в дочерний процесс и оба процесса начинают выполняться параллельно. Важным отличием родительского процесса от дочернего процесса является значение результата, возвращаемого функцией fork(). Дочернему процессу возвращается значение 0, а родительскому процессу возвращается идентификатор дочернего процесса, т.е.:

pid\_t pid = fork();

if (pid == 0) {

//дочерний процесс

}else{

//родительский процесс;

}

где pid – возвращаемое значение, 0 – дочернему процессу, > 0 – родительскому процессу, -1 – в случае ошибки.

Наиболее распространенной схемой выполнения пары процессов (родительский – дочерний), является схема, при которой родительский процесс приостанавливает свое выполнение до завершения дочернего процесса с помощью специальной функции:

pid\_t waitpid(pid\_t pid, int \*status, int options),

где:

pid – идентификатор дочернего процесса, завершение которого ожидается,

status – результат завершения дочернего процесса,

options – режим работы функции.

В некоторых случаях вызов fork() используется программистом для организации параллельного выполнения процессов в рамках одной написанной программы.

В других случаях в качестве дочернего процесса необходимо выполнить внешнюю программу.

В этом случае для запуска внешней программы следует в дочернем процессе вызвать функцию семейства exec().

Существуют следующие разновидности этой функции:

1. int execl(const char \*path, const char \*arg, ...),

2. int execlp(const char \*file, const char \*arg, ...),

3. int execle(const char \*path, const char \*arg,..., char \* const envp[]),

4. int execlpe(const char \*file, const char \*arg , ..., NULL, char \* const envp[]),

5. int execv(const char \*path, char \*const argv[]),

6. int execvp(const char \*file, char \*const argv[]),

7. int execve(const char \* path, char \*const argv[],char \*const envp[]),

8. int execvpe(const char \*file, char \*const argv[],char \*const envp[]).

Если в имени функции присутствует символ ‘l’, то аргументы arg командной строки передаются в виде списка arg0, arg1.... argn, NULL.

Если в имени функции присутствует символ ‘v’, то аргументы командной строки передаются в виде массива argv[]. Отдельные аргументы адресуются через argv[0], argv[1], ..., argv[n]. Последний аргумент (argv [n]) должен быть NULL.

Если в имени функции присутствует символ ‘e’, то последним аргументом функции является массив переменных среды envp[].

Если в имени функции присутствует символ ‘p’, то программа с именем file ищется не только в текущем каталоге, но и в каталогах, определенных переменной среды PATH.

Если в имени функции отсутствует символ ‘p’, то программа с именем path ищется только в текущем каталоге, или имя path должно указывать полный путь к файлу.

Функция execve(), является основной в семействе, остальные функции обеспечивают интерфейс к ней.

В случае успешного выполнения вызова функция не возвращает никакого результата. В случае ошибки возвращается -1, а глобальной переменной errno присваивается значение в соответствии с видом ошибки.

Указания к выполнению работы

Написать программу 1, которая при запуске принимает несколько (3 – 5) аргументов в командной строке, а затем в цикле выводит каждый аргумент на экран с задержкой в несколько секунд.

Программа 1 должна выводить на экран свой идентификатор и идентификатор процесса-родителя.

Программа 1 должна сформировать код завершения.

Написать программу 2, которая запускает программу 1 в качестве дочернего процесса с помощью вызовов fork() и exec().

Программа 2 должна вывести на экран идентификатор процесса-родителя, свой идентификатор и идентификатор дочернего процесса.

Программа 2 должна сформировать набор параметров для передачи в дочерний процесс аргументов командной строки.

Программа 2 должна ожидать завершения дочернего процесса, проверяя событие завершения каждую половину секунды, а по завершению дочернего процесса вывести на экран код завершения.

Студенты с номерами 1, 9 используют 1-й вариант функции exec(). Студенты с номерами 2, 10 используют 2-й вариант функции exec(). И т.д.

*Вывод:*

Мы познакомились с основными системными вызовами, обеспечивающими создание процессов. А также реализовали функционирующее приложение, выполняющее заданные функции.

Приложение 1

Листинг кода

Lab5\_1.cpp

*//*

*// main.cpp*

*// OS\_lab5.1*

*//*

*// Created by MacBook Pro on 01/04/2019.*

*// Copyright © 2019 MacBook Pro. All rights reserved.*

*//*

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

**int** main(**int** argc, **char** \*argv[])

{

std::cout << "Program started"<<std::endl;

pid\_t pid;

**int** status = 0;

**switch**(pid=fork()) {

**case** -1:

perror("fork");

exit(1);

**case** 0:

execvp("./OS\_lab5.2", argv+1);

**default**:

std::cout << "Main process started"<<std::endl;

waitpid(pid, &status, 0);

std::cout<<"Main parent pid "<<getppid()<<std::endl;

std::cout<<"PARENT pid "<<getpid()<<std::endl;

std::cout<<"Child pid "<<pid<<std::endl;

std::cout<<"Child status from parent is "<<status<<std::endl;

std::cout << "WEXITSTATUS(status) = " << WEXITSTATUS(status) << " Child returned this code" << std::endl;

std::cout << "Main process finished"<<std::endl;

}

std::cout << "Program finished"<<std::endl;

**return** 0;

}

Lab5\_2.cpp

*//*

*// main.cpp*

*// OS\_lab5.2*

*//*

*// Created by MacBook Pro on 01/04/2019.*

*// Copyright © 2019 MacBook Pro. All rights reserved.*

*//*

#include <iostream>

#include <unistd.h>

**int** main(**int** argc, **char** \*argv[])

{

std::cout << "Child process started"<<std::endl;

**for** (**int** count = 0; count < argc; ++count)

{

std::cout << count + 1 << " " << argv[count] << '\n';

sleep(1);

}

std::cout << " CHILD: My PID -- " << getpid() << std::endl;

std::cout << " CHILD: Parent PID -- " << getppid() << std::endl;

std::cout << "Child process finished" << std::endl;

exit(**int**(5));

}

Приложение 2

Пример работы программы

