***Лабораторна робота №19***

***«Робота з процесами Linux програмним шляхом мовою C. Системний виклик fork()»***

***Теоретичні відомості***

В ОС Linux для створення процесів використовується системний виклик ***fork()***:

***#include <sys/types.h>***

***#include <unistd.h>***

***pid\_t fork (void);***

В результаті успішного виклику ***fork()*** ядро ​​створює новий процес, який є майже точною копією процесу, який його викликає. Іншими словами, новий процес виконує копію тієї ж програми, що і процес, який створив його, при цьому всі його об'єкти даних мають ті ж самі значення, що і в процесі, що викликає. Створений процес називається **дочірнім процесом**, а процес, який здійснив виклик ***fork()***, називається **батьківським**. Після виклику батьківський процес і його новостворений нащадок виконуються одночасно, при цьому обидва процеси продовжують виконання з оператора, який слід відразу ж за викликом ***fork()***. Процеси виконуються в різних адресних просторах, тому прямий доступ до змінних одного процесу з іншого процесу неможливий.

**Приклад.** Наступна коротка програма більш наочно показує роботу виклику ***fork()*** і використання процесу:

***#include <stdio.h>***

***#include <unistd.h>***

***int main ()***

***{***

***pid\_t pid;*** /\*ідентифікатор процесу\*/

***printf (“Поки всього один процес\n”);***

***pid = fork();*** /\*Створення нового процесу\*/

***printf (“Вже два процеси\n”);***

***if (pid == 0)***

***{***

***printf(“Це дочірній процес, його pid=%d\n”, getpid());***

***printf(“А pid його батьківського процесу=%d\n”, getppid());***

***}***

***else if (pid > 0)***

***printf (“Це батьківський процес, його pid=%d\n”, getpid());***

***else***

***printf (“Помилка виклику fork, нащадок не створений\n”);***

***}***

Для коректного завершення дочірнього процесу в батьківському процесі необхідно використовувати функцію ***wait()*** або ***waitpid()***:

***pid\_t wait(int \*status);***

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

Функція ***wait()*** призупиняє виконання батьківського процесу до тих пір, поки дочірній процес не припинить виконання або до появи сигналу, який або завершує поточний процес, або вимагає викликати функцію-обробник. Якщо дочірній процес до моменту виклику функції вже завершився (так званий «зомбі»), то функція негайно повертається. Системні ресурси, пов'язані з дочірнім процесом, звільняються.

Функція ***waitpid()*** призупиняє виконання батьківського процесу до тих пір, поки дочірній процес, зазначений в параметрі ***pid***, що не завершить виконання, або поки не з'явиться сигнал, який або завершує батьківський процес, або вимагає викликати функцію-обробник. Якщо вказаний дочірній процес до моменту виклику функції вже завершився (так званий «зомбі»), то функція негайно повертається. Системні ресурси, пов'язані з дочірнім процесом, звільняються. Параметр ***pid*** може приймати кілька значень:

* **pid<-1** означає, що потрібно чекати будь-якого дочірнього процесу, чий ідентифікатор групи процесів дорівнює абсолютному значенню ***pid***;
* **pid=-1** означає очікувати будь-якого дочірнього процесу; функція ***wait()*** поводиться точно так само;
* **pid=0** означає очікувати будь-якого дочірнього процесу, чий ідентифікатор групи процесів дорівнює такому у пов'язаних з поточною діяльністю;
* **pid>0** означає очікувати дочірнього процесу, чий ідентифікатор дорівнює ***pid***.

Значення ***options*** створюється шляхом бітової операції АБО над наступними константами:

* ***WNOHANG*** - означає повернути управління негайно, якщо жоден дочірній процес не завершив виконання.
* ***WUNTRACED*** - означає повертати управління також для зупинених дочірніх процесів, про чий статус ще не було повідомлено.

Кожний дочірній процес при завершенні роботи посилає свому процесу-батькові спеціальний сигнал ***SIGCHLD***, на який у всіх процесів за замовчуванням встановлена реакція "ігнорувати сигнал". Наявність такого сигналу спільно з системним викликом ***waitpid()*** дозволяє організувати асинхронний збір інформації про статус завершених породжених процесів процесом-батьком.

Для перевантаження виконуваної програми можна використовувати функції сімейства ***exec***. Основна відмінність між різними функціями в сімействі полягає в способі передачі параметрів.

***int execl(char \*pathname, char \*arg0, arg1, ..., argn, NULL);***

***int execle(char \*pathname, char \*arg0, arg1, ..., argn, NULL, char \*\*envp);***

***int execlp(char \*pathname, char \*arg0, arg1, ..., argn, NULL);***

***int execlpe(char \*pathname, char \*arg0, arg1, ..., argn, NULL, char \*\*envp);***

***int execv(char \*pathname, char \*argv[]);***

***int execve(char \*pathname, char \*argv[],char \*\*envp);***

***int execvp(char \*pathname, char \*argv[]);***

***int execvpe(char \*pathname, char \*argv[],char \*\*envp);***

Основна відмінність між різними функціями в сімействі полягає в способі передачі параметрів. Як видно з рис. 1, всі ці функції виконують один системний виклик ***execve***.

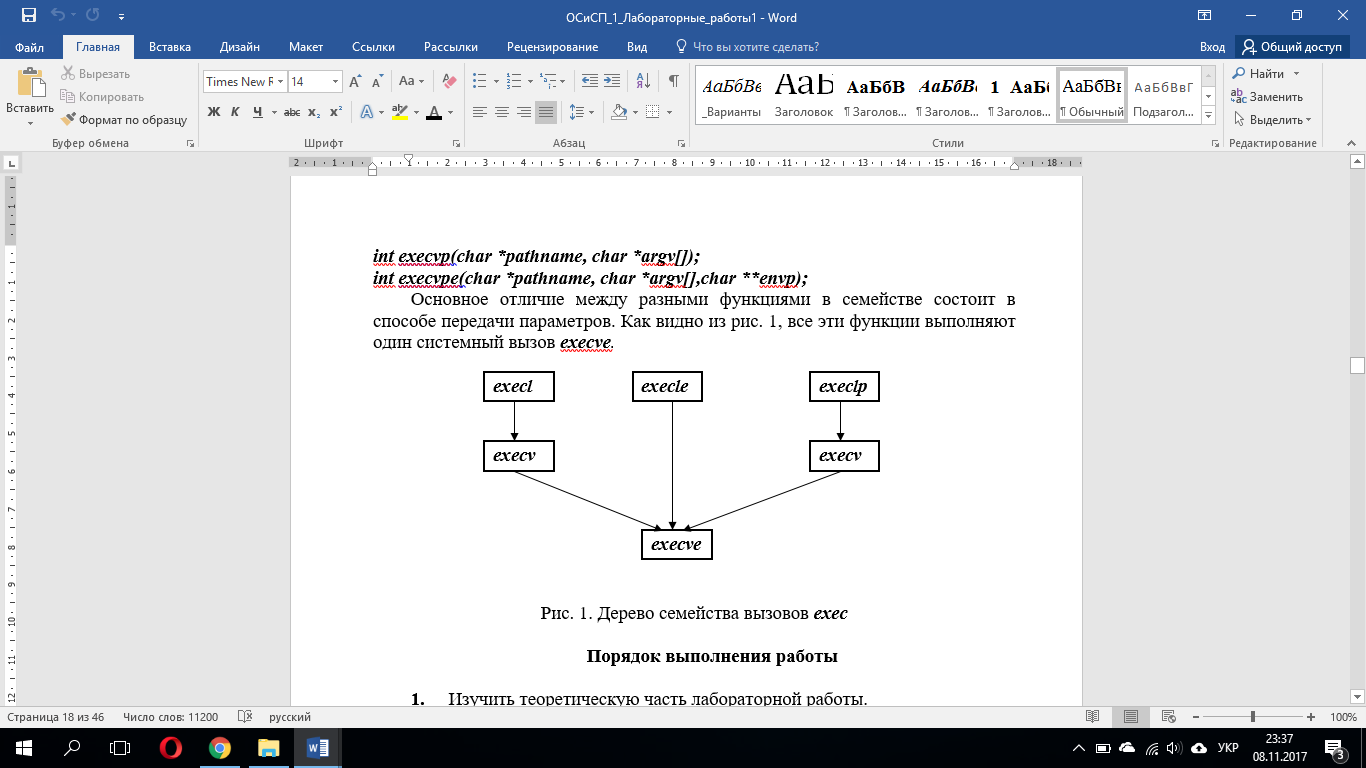


Рис. 1. Дерево сімейства викликів ***exec***

***Завдання***

1. Напишіть програму ***prog.c*** відповідно до схеми, представленої на рис. 2.

Батьківський процес відкриває існуючий текстовий файл 1 для читання; потім створює дочірній процес. Після цього батьківський процес створює новий файл 2 для запису і копіює в нього вміст файлу 1 блоками по 10 байтів кожний.

Одночасно з цим дочірній процес створює новий файл 3 для запису і копіює в нього вміст файлу 1 блоками по 20 байтів кожний.

Після завершення копіювання обидва процеси завершуються. Програма повинна запускатися з трьома параметрами:

***$prog file1 file2 file3***



1. Запустіть програму кілька разів для заданого значення ***N*** з одними і тими ж іменами файлів і порівняйте розмір і вміст файлів 2 і 3 з розміром і вмістом файлу 1. Поясніть результати.
2. Повторіть попередній пункт для ***N = 10***, ***5000***, ***10000*** та ***100000***.

***Контрольні питання***

1. Які існують способи програмно запустити новий процес? Чи вони відрізняються один від одного?
2. Які функції входять до сімейства ***exec***?
3. Що відбувається при системному виклику ***fork()***?
4. Як програмним чином завершити процес?
5. Яка функція призначена для очікування завершення дочірніх процесів?
6. Чим функція ***waitpid()*** відрізняється від функції ***wait()***?