Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

ОПЕРАЦІЙНІ СИСТЕМИ

Комп’ютерний практикум

Робота № 6

Виконав
студент гр. ФЕ-01 Дорошенко В. О.
Перевірив
Кіреєнко О. В.

Київ - 2022

Застосуванням системних викликів fork() і exec()

Мета

Оволодіння практичними навичками застосування системних викликів у програмах, дослідження механізму створення процесів у UNIX-подібних

Варіант 6 Залікова книжка ФЕ-0108

Інформація о системі

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64
```

1.

```cpp
#include <iostream>
#include <string>
// Required by for routine
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h> // Declaration for exit()
using namespace std;
int globalVariable = 2;
int main()
{
        string sIdentifier;
        int iStackVariable = 20;

        pid_t pID = fork();
        if (pID == 0) // child
        {
        // Code only executed by child process

        sIdentifier = "Child Process: ";
        globalVariable++;
        iStackVariable++;
        }
        else if (pID < 0) // failed to fork
        {
        cerr << "Failed to fork" << endl;
        exit(1);
        // Throw exception
        }
        else // parent
        {
        // Code only executed by parent process

        sIdentifier = "Parent Process:";
        }

        // Code executed by both parent and child.

        cout << sIdentifier;
        cout << " Global variable: " << globalVariable;
        cout << " Stack variable: " << iStackVariable << endl;
}
```

2. Скомпілюйте програму (вважаємо, текст збережено у файлі myforktest.cpp)

```
→  ~ g++ -o myforktest myforktest.cpp
```

3. Запустіть програму myforktest. У якій послідовності виконуються батьківський процес і процес-нащадок? Чи завжди цей порядок дотримується?

```
→  ~ ./myforktest
Parent Process: Global variable: 2 Stack variable: 20
Child Process:  Global variable: 3 Stack variable: 21
→  ~ ./myforktest
Child Process:  Global variable: 3 Stack variable: 21
Parent Process: Global variable: 2 Stack variable: 20
```

Батьківський та дочірній процеси працюють ”паралельно”, але оскільки в дочірньому процесі є операції інкрементування, то більш ймовірно, що він виконається пізніше батьківського (бо витратить час на інкрементування) "Паралельно" - в тому сенсі що хоч і планувальник ставить процеси виконуються по черзі, але виконання може перерватися після будь-якої атомарної операції, як тільки закінчиться виділений процесорний час. Якщо добавити sleep(1) процесу-нащадку, то завжди буде виконуватись першим батьківський процес

```
  sIdentifier = "Child Process: ";
  globalVariable++;
  iStackVariable++;
  sleep(1);
```

```
→  ~ ./myforktest
Parent Process: Global variable: 2 Stack variable: 20
→  ~ ./myforktestChild Process:  Global variable: 3 Stack variable:

Parent Process: Global variable: 2 Stack variable: 20
→  ~ Child Process:  Global variable: 3 Stack variable: 21
./myforktest
Parent Process: Global variable: 2 Stack variable: 20
→  ~ Child Process:  Global variable: 3 Stack variable: 21
```

4. Додайте затримку у виконання одного або обох з цих процесів (функція sleep(), аргумент — затримка у секундах). Чи змінились результати виконання?

```
→  ~ ./myforktest
Child Process:  Global variable: 3 Stack variable: 21
Parent Process: Global variable: 2 Stack variable: 20
→  ~
```

Тепер першим встигатиме виконуватися дочірній процес до батьківського

5. Додайте цикл, який забезпечить кількаразове повторення дій після виклику fork(). Які результати показують процеси (значення глобальної змінної і змінної, що визначена у стеку)? Поясніть.

```
        pid_t pID = fork();
        for (int i=0; i<3; ++i)
        {
                if (pID == 0) // child
                {
                // Code only executed by child process

                sIdentifier = "Child Process: ";
                globalVariable++;
                iStackVariable++;
                }
                else if (pID < 0) // failed to fork
                {
                cerr << "Failed to fork" << endl;
                exit(1);
                // Throw exception
                }
                else // parent
                {
                // Code only executed by parent process

                sIdentifier = "Parent Process:";
                sleep(3);
                }
                // Code executed by both parent and child.

                cout << sIdentifier;
                cout << " Global variable: " << globalVariable;
                cout << " Stack variable: " << iStackVariable << endl;
        }
}
```

```
→  ~ ./myforktest
Child Process:  Global variable: 3 Stack variable: 21
Child Process:  Global variable: 4 Stack variable: 22
Child Process:  Global variable: 5 Stack variable: 23
Parent Process: Global variable: 2 Stack variable: 20
Parent Process: Global variable: 2 Stack variable: 20
Parent Process: Global variable: 2 Stack variable: 20
```

При fork() дочірні процеси копіюють адресний простір батьківського процессу, а не шарять з ним одну пам'ять. Тому значення змінних в батьківському процесі не змінилось.

6. Спробуйте у первинній програмі (без циклу) замість виклику fork() здійснити виклик vfork().

```
→  ~ ./myforktest
Child Process:  Global variable: 3 Stack variable: 21
[1]    9103 segmentation fault (core dumped)  ./myforktest
```

При vfork() адресний простір не копіюється, а батьківський процес припиняється до завершення процесу-нащадка. Виникає помилка segmentation fault, оскільки до тих пір, поки процес-нащадок не виконає успішне виконання exec(), або не викличе _exit() батьківський процес не може виконуватися.

7. . Тепер додайте виклик exec() у код процесу-нащадка. Для початку використайте простішу функцію execl(). Варіант виклику на прикладі утиліти ls: execl("/bin/ls", "/bin/ls", "-a", "-l", (char *) 0);

```
    globalVariable++;
    iStackVariable++;
    execl("/bin/ls", "/bin/ls", "-a", "-l", (char *) 0);
    }
    else if (pID < 0) // failed to fork
```

```
→  ~ g++ -o myforktest myforktest.cpp
→  ~ ./myforktest
total 195112
drwx------. 21 vl   vl       4096 May 12 22:02 .
drwxr-xr-x.  4 root root       28 Feb 18 05:38 ..
-rw-rw-r--.  1 vl   vl          0 Mar  7 20:30 1
-rwxrwxr-x.  1 vl   vl      13880 May 12 20:17 a.out
-rw-------.  1 vl   vl       8041 Mar  6 18:41 .bash_history
-rw-r--r--.  1 vl   vl         18 Apr  1  2020 .bash_logout
-rw-r--r--.  1 vl   vl        193 Apr  1  2020 .bash_profile
-rw-r--r--.  1 vl   vl        230 Mar 16 15:24 .bashrc
drwx------. 16 vl   vl       4096 May 12 19:45 .cache
drwxr-xr-x. 16 vl   vl       4096 Feb  9 13:48 .config
drwx------.  3 vl   vl         25 Feb  8 12:04 .dbus
-rw-r--r--.  1 vl   vl       3882 Mar 16 15:26 .zshrc
Parent Process: Global variable: 3 Stack variable: 21
```

8. Проведіть експерименти з викликом різних програм, у тому числі ps, bash, а також з викликами execl() у батьківському процесі.

```
  iStackVariable++;
  execl("/bin/ps", "/bin/ps", "ufx", (char *) 0);
  }
```

```
→  ~ vim myforktest.cpp
→  ~ g++ -o myforktest myforktest.cpp
→  ~ ./myforktest
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
vl        6972  0.0  0.0 161072  1636 ?        S    20:02   0:01 sshd: vl@pts/2
vl        6975  0.0  0.1 149792  3996 pts/2    Ss   20:02   0:01  \_ -zsh
vl        9797  0.0  0.0  12676   880 pts/2    S+   22:08   0:00      \_ ./myforktest
vl        9798  0.0  0.0 155588  1852 pts/2    R+   22:08   0:00          \_ /bin/ps ufx
vl        4655  0.0  0.0 149636  2212 tty4     Ss+  19:47   0:01 -zsh
vl        4526  0.0  0.0 147116  1308 tty3     Ss+  19:46   0:00 -zsh
```

```
    iStackVariable++;
    execl("/bin/zsh", "/bin/zsh", "/home/vl/mytest.sh", (char *) 0);
    }
```

```
→  ~ g++ -o myforktest myforktest.cpp
→  ~ ./myforktest
vl
Parent Process: Global variable: 3 Stack variable: 21
```

Як запустити фоновий процеснащадок?

В exec() останнім аргументом має бути “&”

Як процес-нащадок дізнається власний PID?
Функція getpid () повертає ідентифікатор процесу поточного процесу.
PID батьківського процесу?
Процес може знайти PID свого батька за допомогою системного виклику getppid().

Висновок:

Отримав розуміння работы системних викликів fork() і exec().

Навчився створювати нові процеси в Unix-подобніх системах.

Ознайомився з помилками які можуть виникнути при створенні процесів.