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

Операційні системи

Лабораторна №6

Виконав:
Студент групи ФБ-82
Козачок Вячеслав
Перевірив:
Кіреєнко О.В.

Київ - 2020

1 Завдання до виконання

1. Для початку можна взяти демонстраційну програму, запропоновану.

```
#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`
   Увага! Пам'ятайте, що не можна називати власні програми просто test! test – це вбудована команда shell (з якою ви вже зустрічалися в Роботі №4).

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

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

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.

6. Спробуйте у первинній програмі (без циклу) замість виклику fork() здійснити виклик vfork(). У чому різниця роботи цих двох викликів? Чи виникає помилка (якщо так, то яка)? У чому причина? Як "змусити" працювати виклик vfork()? Які результати тепер показують процеси (значення глобальної змінної і змінної, що визначена у стеку)? Поясніть.

7. Тепер додайте виклик exec() у код процесу-нащадка. Для початку використайте простішу функцію execl(). Варіант виклику на прикладі утиліти ls: execl(або"/bin/ls "/bin/ls a l (абоchar *) 0); У наведеному прикладі передаються аргументи командного рядка.

8. Проведіть експерименти з викликом різних програм, у тому числі ps, bash, а також з викликами execl() у батьківському процесі. Як запустити фоновий процес-нащадок? Як процес-нащадок дізнається власний PID? PID батьківського процесу?

9. Усі отримані результати і відповіді на запитання, які були задані вище, іть у вигляді протоколу.

Виконання роботи

Task 2

```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;
    pID = fork();
    if (pID == 0)
    {
        // Code only executed by child process
        sIdentifier = "Child Process: ";
        globalVariable++;
        iStackVariable++;
    }
    else if (pID < 0)
    {
        cerr << "Failed to fork" << endl;
        exit(1);
        // Throw exception
    }
    else
    {
        // Code only executed by parent process
        sIdentifier = "Parent Process:";
        sleep(1);
    }
    // Code executed by both parent and child.
    cout << sIdentifier;
    cout << " Global variable: " << globalVariable;
    cout << " Stack variable: " << iStackVariable
        << " Pid: " << pID << endl;
}
```

Terminal:

```
g++ -g main.cpp -o main
./main
Child Process:  Global variable: 3 Stack variable: 21 Pid: 0
Parent Process: Global variable: 2 Stack variable: 20 Pid: 20372
```

Task 3

Послідовність не змінюється. Через деяку кількість викликів не змінилась послідовність. Спочатку відпрацьовує батьківський процесс, а вже потім дочірній.

Task 4

```cpp
...
else
{
// Code only executed by parent process
sIdentifier = "Parent Process:";
sleep(1);
}
...
```

Terminal:

```
g++ -g main.cpp -o main
./main
Child Process:  Global variable: 3 Stack variable: 21 Pid: 0
Parent Process: Global variable: 2 Stack variable: 20 Pid: 20625
```

Якщо додати `sleep` в батьківську частину коду, то спочатку відпрацьовує дочірня частина, а потім батьківська, проте, якщо додати в дочірню частину, то спочатку виводить батьківську частину, дозволяє вводити команди в `shell` проте программа дитячого процесу продовжує вивід у оболочку.

```
g++ -g main.cpp -o main
./main
Parent Process: Global variable: 2 Stack variable: 20 Pid: 21091
[eski@eski-pc Lab6]$ Child Process:  Global variable: 3 Stack variable: 21 Pid: 0
```

Task 5

```
int main()
{
    string sIdentifier;
    int iStackVariable = 20;
    pid_t pID;
    pID = fork();
    for(int i = 0; i < 4; i++)
    {
        if (pID == 0)
        {
            // Code only executed by child process
            sIdentifier = "Child Process: ";
            globalVariable++;
            iStackVariable++;
        }
        else if (pID < 0)
        {
            cerr << "Failed to fork" << endl;
            exit(1);
            // Throw exception
        }
        else
        {
            // Code only executed by parent process
            sIdentifier = "Parent Process:";
            sleep(1);
        }
        // Code executed by both parent and child.
        cout << sIdentifier;
        cout << " Global variable: " << globalVariable;
        cout << " Stack variable: " << iStackVariable
            << " Pid: " << pID << endl;
    }
}
```

Terminal:

```
g++ -g main.cpp -o main
./main
Child Process:  Global variable: 3 Stack variable: 21 Pid: 0
Child Process:  Global variable: 4 Stack variable: 22 Pid: 0
Child Process:  Global variable: 5 Stack variable: 23 Pid: 0
Child Process:  Global variable: 6 Stack variable: 24 Pid: 0
Parent Process: Global variable: 2 Stack variable: 20 Pid: 20848
Parent Process: Global variable: 2 Stack variable: 20 Pid: 20848
Parent Process: Global variable: 2 Stack variable: 20 Pid: 20848
Parent Process: Global variable: 2 Stack variable: 20 Pid: 20848
```

Якщо поставити sleep(1) в частину батьківську, то кожна стрічка “Parent process” виводиться через секунду після попередньої.

Task 6

```
...
pid_t pID;
pID = vfork();
if (pID == 0)
{
...
```

Terminal:

```
g++ -g main.cpp -o main
./main
Child Process:  Global variable: 3 Stack variable: 21 Pid: 0
Parent Process: Global variable: 3 Stack variable: 21 Pid: 21255
*** stack smashing detected ***: terminated
make: *** [Makefile:3: c] Aborted (core dumped)
```

`vfork()` не створює копію батьківського процесу, а створює поділюваний з батьківським процессом адресний простір до тих пір, поки не буде викликана функція `_exit` чи одна з функцій сімейства `exec`

Щоб не ставалося помилки, треба закрити дочірній процес викликаний викликом`vfork()`.

Але оскільки ми отримуємо адресний простір батьківського процесу, то дочірній процес може змінювати змінні батьківського, отже глобальна змінна, як і зміна стеку, зміняться.

```
...
 if (pID == 0)
{
    // Code only executed by child process
    sIdentifier = "Child Process: ";
    sleep(1);
    globalVariable++;
    iStackVariable++;
    exit(EXIT_SUCCESS);
}
...
```

Terminal:

```
g++ -g main.cpp -o main
./main
Parent Process: Global variable: 3 Stack variable: 21 Pid: 23049
```

Проте як ми можемо бачити, то процес дочірній завершився та не вивів тексту у стандартний потік.

Task 7

```
 if (pID == 0)
{
    // Code only executed by child process
    sIdentifier = "Child Process: ";
    sleep(1);
    globalVariable++;
    iStackVariable++;
    execl("/bin/ls", "-a", "-l", 0);
    _exit(EXIT_SUCCESS);
}
```

Terminal:

```
[eski@eski-pc Lab6]$ make
g++ -g main.cpp -o main
./main
Parent Process: Global variable: 2 Stack variable: 20 Pid: 24651
[eski@eski-pc Lab6]$ total 216
drwxr-xr-x 3 eski eski  4096 Mar 19 22:05 .
drwxr-xr-x 8 eski eski  4096 Mar 19 17:42 ..
-rw-r--r-- 1 eski eski   879 Mar 19 22:02 lab4.aux
-rw-r--r-- 1 eski eski 28894 Mar 19 22:02 lab4.log
-rw-r--r-- 1 eski eski   211 Mar 19 22:02 lab4.out
-rwxr-xr-x 1 eski eski 74960 Mar 19 22:05 main
-rw-r--r-- 1 eski eski  1025 Mar 19 22:04 main.cpp
-rwxr-xr-x 1 eski eski 74744 Mar 19 19:40 main.o
-rw-r--r-- 1 eski eski    46 Mar 19 19:45 Makefile
drwxr-xr-x 2 eski eski  4096 Mar 19 22:04 tex
-rw-r--r-- 1 eski eski   626 Mar 19 22:00 title.aux
```

Використання ps:

```
if (pID == 0)
{
    // Code only executed by child process
    sIdentifier = "Child Process: ";
    sleep(1);
    globalVariable++;
    iStackVariable++;
 -> execl("/bin/ps", "/bin/ps", "-U", "eski", 0);
    _exit(EXIT_SUCCESS);
}
```

```
[eski@eski-pc Lab6]$ make
g++ -g main.cpp -o main
./main
Parent Process: Global variable: 2 Stack variable: 20 Pid: 25397
[eski@eski-pc Lab6]$     PID TTY          TIME CMD
1506 ?        00:00:00 systemd
1507 ?        00:00:00 (sd-pam)
1518 ?        00:00:04 kwalletd5
1519 ?        00:00:00 startplasma-x11
1529 ?        00:00:19 dbus-daemon
1543 ?        00:00:00 start_kdeinit
1544 ?        00:00:00 kdeinit5
1560 ?        00:00:05 klauncher
```

Task 8

- Ми можемо запустити фоновий процес-нащадок використовуючи: `waitpid()`
- `fork()` повертає дочірній PID батьківському процессу.
- Процес-нащадок може знайти PID батьківському процессу використовуючи `getppid()`.

Висновки

В цій роботі я зрозумів, як у ОС UNIX здійснюється створення дочірніх процесів батьківським процесом. Зрозумів, коли процеси користуються спільним адресним простором, а коли для кожного з них виділяється свій адресний простір.