HỆ ĐIỀU HÀNH
TUẦN 5
TẠO TIẾN TRÌNH

Hướng dẫn làm bài

Định nghĩa của tiến trình: là một thực thể điều khiển đoạn mã lệnh có riêng một không gian địa chỉ, có ngăn xếp stack riêng rẽ, có bảng chứa các thông số mô tả file được mở cùng tiến trình và đặc biệt có một định danh PID (Process Identify) duy nhất trong toàn bộ hệ thống vào thời điểm tiến trình đang chạy.

Như chúng ta đã biết, tiến trình không phải là một chương trình (tuy đôi lúc một chương trình đơn giản chỉ cần một tiến trình duy nhất để hoàn thành tác vụ, trong trường hợp này thì chúng ta có thể xem tiến trình và chương trình là một).

Một tiến trình cũng trải qua các quá trình như con người: Nó được sinh ra, nó có thể có một cuộc đời ít ý nghĩa hoặc nhiều ý nghĩa, nó có thể sinh ra một hoặc nhiều tiến trình con, và thậm chí, nó có có thể chết đi. Điều khác biệt nhỏ duy nhất là: tiến trình không có giới tính. Mỗi tiến trình chỉ có một tiến trình cha (hoặc có thể gọi là mẹ, ở trong khoa học sẽ thống nhất gọi là cha) duy nhất.

Dưới góc nhìn của kernel, tiến trình là một thực thể chiếm dụng tài nguyên của hệ thống (Thời gian sử dụng CPU, bộ nhớ, …)

Khi một tiến trình con được tạo ra, nó hầu như giống hệt như tiến trình cha. Tiến trình con sao chép toàn bộ không gian địa chỉ, thực thi cùng một mã nguồn như tiến trình cha, và bắt đầu chạy tiếp những mã nguồn riêng cho tiến trình con từ thời điểm gọi hàm tạo tiến trình mới.

Mặc dù tiến trình cha và tiến trình con cùng chia sẻ sử dụng phần mã nguồn của chương trình, nhưng chúng lại có phần dữ liệu tách biệt nhau (stack và heap). Điều này có nghĩa là, những sự thay đổi dữ liệu của tiến trình con không ảnh hưởng đến dữ liệu trong tiến trình cha. Để có thể quản lý được các tiến trình, ta phải sử dụng thư viện:

```
#include <unistd.h>
```

Để thực hiện kiểm soát các tiến trình, ta có thể sử dụng câu lệnh:

- Lấy ID của tiến trình hiện hành `pid_t getpid(void)`
- Lấy ID của tiến trình cha `pid_t getppid(void)`

Ví dụ 1. Giả sử có file **p1.c** chứa hàm main như sau:

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
        printf("current process ID: %d\n", getpid());
        printf("Parent process ID: %d\n", getppid());
        return 0;
}
```

Khi thực thi ta sẽ có kết quả như sau:

```
vunguyen@vunguyen:~/lab3/f1$ gcc -c p1.c
vunguyen@vunguyen:~/lab3/f1$ gcc -o p1.out p1.o
vunguyen@vunguyen:~/lab3/f1$ ./p1.out
current process ID: 6328
Parent process ID: 6302
```

Trong đó số 6328 là chỉ số ID của tiến trình hiện hành, ID của tiến trình cha của tiến trình hiện hành là 6302.

Mỗi tiến trình cũng có thể sinh ra tiến trình con của nó. Để sinh ra tiến trình con, ta phải sử dụng các hàm tạo tiến trình như sau:

Hàm tạo tiến trình: **pid_t fork(void)**

pid_t pid=fork()

Nếu thành công:

pid = 0: trong thân process con

pid > 0: xử lý trong thân process cha

Nếu thất bại: **pid = -1** kèm lý do

ENOMEM: không đủ bộ nhớ

EAGAIN: số tiến trình vượt quá giới hạn cho phép

Ví dụ:

```c
pid_t pid = fork();
if(pid==0){ Thân chương trình con ở đây }
else if (pid>0) { Thân chương trình cha ở đây }
     else { Lỗi ở đây }
```

Ví dụ 2. Giả sử có file **p2.c** chứa hàm main như sau:

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    int pid = fork();
    if (pid == 0)
    {
        printf("I'm the child, my id is %i\n",getpid());
    }
    else
    {
        printf("I'm the parent %i, my child is %i\n",getpid(), pid);
        // here we can kill the child, but that's not very parently of us
    }
    return 0;
}
```

Mỗi tiến trình có thể quản lý một phần việc riêng rẽ với nhau. Khi ta muốn chủ động kết thúc một tiến trình ta có thể dùng hàm: **exit(0);**

Trong một số trường hợp, tiến trình cha cần chờ tiến trình con kết thúc trước khi tiếp tục thực hiện công việc của nó. Các ***system calls*** như **wait, waitpid** được xây dựng để phục vụ việc này.

Hàm **wait**, cho phép tiến trình cha (tiến trình hiện tại) chờ (tạm ngưng tiến trình hiện tại ngay tại lời gọi hàm wait) cho đến khi bất kỳ một tiến trình con nào của nó kết thúc.

Nguyên mẫu của hàm này như sau: **pid_t wait(int *status);**

Ví dụ 3 sau đây tạo ra một số lượng tiến trình con (được truyền vào qua đối số hàm **main**), in ra ID của các tiến trình con và lời gọi chờ của tiến trình cha.

```c
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    int pnum, count, retval, child_no;
    pnum = atoi(argv[1]);
    if(pnum <= 0)
    {
        printf("So tien trinh con phai lon hon 0.”);
        return -1;
    }
    else
```

```
    {
        retval=1;
        for(count=0, count<pnum; count++)
        {
            if(retval!=0)
                retval=fork();
            else break;
        }
        if(retval == 0)
        {
            child_no = count;
            printf("Tien trinh %d, PID %d\n",child_no, getpid());
        }
        else
        {
            for(count=0; count<pnum; count++) wait(NULL);
            printf("Tien trinh cha PID %d", getpid());
        }
    }
    return 0;
}
```

Bài tập thực hành

Bài 1:

Hãy viết 1 chương trình: xuất ra ID của tiến trình hiện tại và ID tiến trình cha của nó. Sau đó, hãy nhân đôi tiến trình này. Tại tiến trình cha và con, hãy xuất ra ID của tiến trình của nó và ID tiến trình cha của nó.

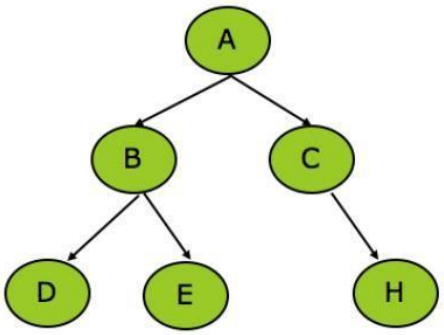
Hãy thêm câu lệnh wait(NULL) vào phần thân của tiến trình cha. Hãy cho biết có điều gì khác biệt.

Bài 2:

Hãy viết 1 hàm để xuất ra thông tin tiến trình hiện tại và tiến trình cha của nó.

Nếu 1 tiến trình sinh ra 1 tiến trình con, hãy xuất thông tin của tiến trình con (đó chính là chỉ số PID).

Hãy tạo cây tiến trình sau:



Bài 3:

Viết chương trình để truyền đối vào số nguyên dương n vào và:

- Tiến trình cha tiếp tục tính rồi xuất ra tổng S = 1 + 2 + …. + n
- Đồng thời tạo một tiến trình con tính tổng các ước số của n và in ra màn hình.

Hãy sử dụng lời gọi wait(). Nếu không sử dụng lời gọi wait() thì rủi ro nào có thể xảy ra cho chương trình này?

HẾT