BÀI 3: QUẢN LÝ TIẾN TRÌNH

I. Mục tiêu:

Hướng dẫn sinh viên thực hiện việc quản lý các tiến trình, bao gồm: tạo ra, quản lý, xóa bỏ các tiến trình. Quản lý các tiến trình cha, tiến trình con. Áp dụng vào các bài toán để giải quyết.

II. Hướng dẫn thực hiện:

Tiến trình là một chương trình đang thực thi, là một thực thể hoạt động, với con trỏ lệnh xác định chỉ thị kế tiếp sẽ thi hành, kèm theo tập các tài nguyên phục vụ cho hoạt động của tiến trình. Mỗi tiến trình sẽ được hệ điều hành gán cho các chỉ số ID tương ứng với hoạt động của nó.

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:

- Giả sử có file p1.c chứa hàm main như sau:

```c
#include <stdio.h>
#include <unistd.h>
void main()
{
        printf ("Current process id %d\n", getpid());
        printf ("Parent process id %d\n", getppid());
}
```

Khi thực thi ta sẽ có kết quả như sau:

```
root@PC:/home/huan/Desktop/bai tap/Process# gcc -c p1.c
root@PC:/home/huan/Desktop/bai tap/Process# gcc -o p1.out p1.o
root@PC:/home/huan/Desktop/bai tap/Process# ./p1.out
Current process id 3525
Parent process id 2710
root@PC:/home/huan/Desktop/bai tap/Process#
```

Hình 3. 1 Minh họa chạy chương trình lấy ProcessID

Trong đó số 3525 là chỉ số ID của tiến trình hiện hành, ID của tiến trình cha của tiến trình hiện hành là 2710.

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:

```
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
}

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, hàm này 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 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.

```
#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;
}
```

III. Bài tập thực hành

Bài 1:

Tạo 1 thư mục có tên là phân số. Trong thư mục này tạo 3 file đặt tên như sau: khaibaops.c, tinhtoanps.c và xulyps.c

Trong file khaibaops.c xây dựng các hàm gồm nhập phân số và xuất phân số.

Trong file tinhtoanps.c xây dựng các hàm cộng, trừ, nhân chia phân số để tính toán phân số.

Từ 2 file khaibaops.c và tinhtoanps.c hãy xây dựng thư viện tĩnh có tên là pstinh.a, xây dựng thư viện động có tên là psdong.a. Chép psdong.a sang thư viện /lib.

Tại file xulyps.c hãy xây dựng 1 cấu trúc phân số. Sau đó sử dụng các hàm có trong bộ thư viện vừa mới xây dựng, hãy cho người dùng nhập phân số. Sau đó lần lượt tính các phép toán cộng, trừ, nhân, chia phân số và xuất kết quả.

Tạo 1 thư mục mới nằm ngoài thư mục phân số có tên là hỗn số. Trong thư mục này tạo file: xulyhonso.c

Trong file xulyhonso.c hãy gọi thư viện đã tạo ở trên. Hãy xây dựng thêm hàm đổi hỗn số sang phân số, sau đó thực hiện các phép tính cộng trừ nhân chia hỗn số này.

Hãy cho biết sự khác biệt khi sử dụng thư viện tĩnh và động trong việc áp dụng qua file xulyhonso.c.

Bài 2:

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.

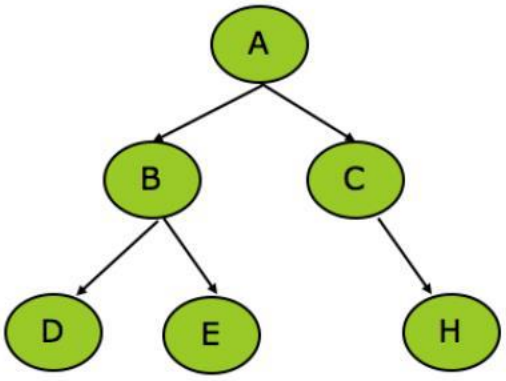
Hãy thêm câu lệnh wait(NULL) vào phần thân của tiến trình cha. Hãy cho biết có điều gì khác biệt.

Bài 3:

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 4:

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() để chắc chắn rằng tiến trình con không trở thành một tiến trình mồ côi. Nếu không sử dụng lời gọi wait() thì rủi ro nào có thể xảy ra cho chương trình này?

Bài 5:

Phỏng đoán Collatz tin rằng dãy số sinh ra sẽ luôn tiến về 1 với bất kỳ số nguyên dương nào được tạo ra ở bước đầu. Dãy số được tạo ra theo giải thuật sau:

$$n = \begin{cases} \frac{n}{2} \; nếu \; n \; là \; số \; chẵn \\ 3 * n + 1 \; nếu \; n \; là \; số \; lẻ \end{cases}$$

Ví dụ với n=35, dãy số sinh ra là 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Viết một chương trình nhận số nguyên dương n vào thông qua đối số, kiểm tra tính đúng của giá trị này. Tạo ra một tiến trình con để tính và in ra dãy số, tronng lúc tiến trình cha cần chờ tiến trình con hoàn thành thông qua lời gọi wait().