Tạo và quản lý tiến trình

# Bài tập 1 :

Xem thông tin về tiến trình, trong Linux ta có thể dùng lệnh ps , top . Trong Windows , ta có công cụ Task Manager.

Sử dụng các công cụ trên để xem thông tin về tiến trình.

**Bài tập 2 : Tạo tiến trình bằng fork() system call**

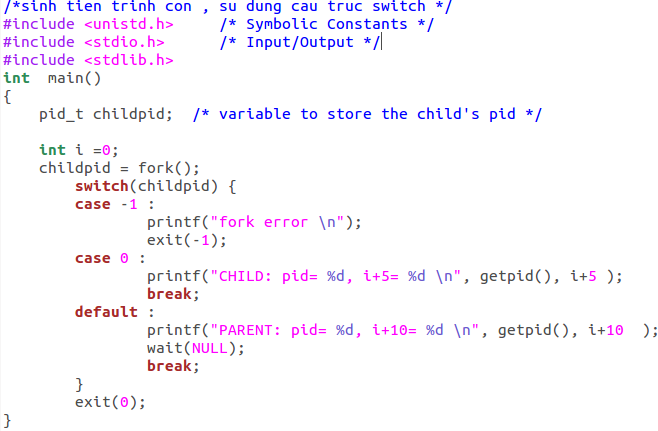
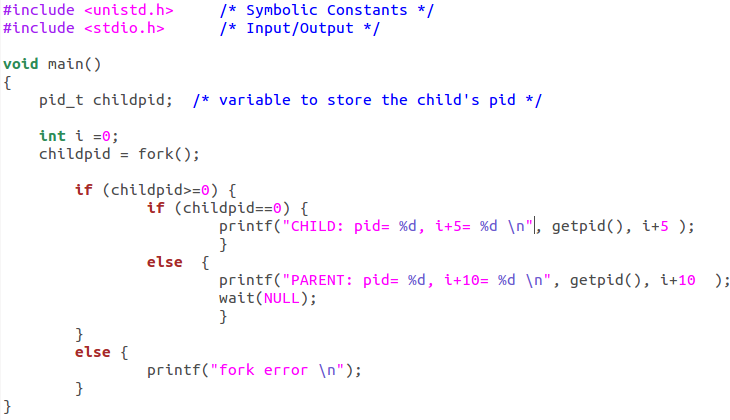
* Fork() system call cho phép một tiến trình (parent) tạo một tiến trình mới (child). Fork() tạo tiến trình con bằng cách sao chép không gian địa chỉ của tiến trình cha : tiến trình con chứa bản copy các stack , data, heap, text segment của tiến trình cha.

Sau khi tiến trình con được tạo, Cả 2 tiến trình sẽ bắt đầu thực thi từ câu lệnh kế tiếp lời gọi fork() .

* Hàm fork() trả về giá trị :
  + nếu tạo tiến trình con không thành công : -1
  + nếu tạo tiến trình con thành công : fork() trả về 0 cho tiến trình con , và trả về PID của tiến trình con cho tiến trình cha

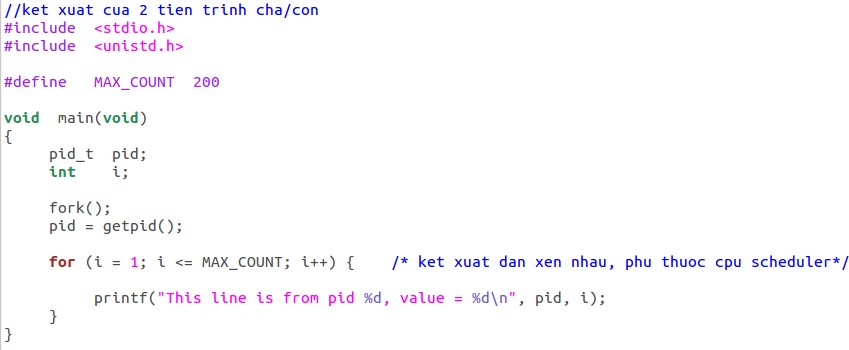
*Tham khảo : man fork*

**Chương trình fork01.c : phân biệt xử lý tiến trình cha-con bằng sử dụng cấu trúc if-else hoặc switch-case . Hãy đọc hiểu chương trình, thực thi và xem kết quả ?**

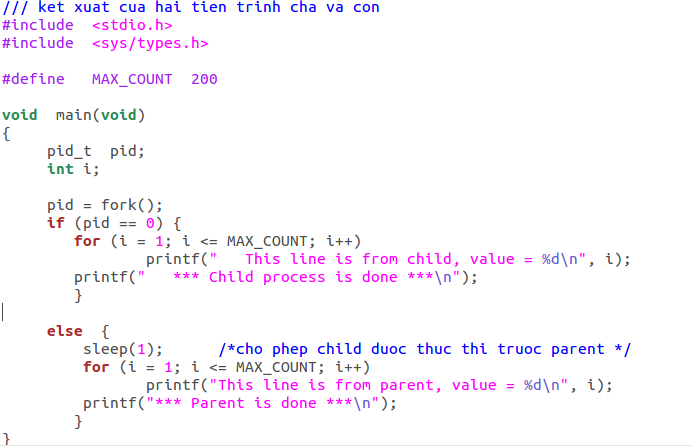


**Bài tập 3 : Tạo tiến trình bằng fork() system call (tiếp theo)**

**Chương trình fork02.c : Kết xuất của hai tiến trình đan xen nhau , phụ thuộc vào bộ lập lịch. Hãy đọc hiểu chương trình, thực thi và xem kết quả ?**



**Hoặc, sử dụng sleep() để cho phép tiến trình con thực thi và có thể hoàn thành tác vụ trước tiến trình cha**:



**Bài tập 4 : Thực thi chương trình**

**execve() system call** nạp một chương trình mới vào vùng nhớ của process hiện hành. Các text, data, stack segment của process hiện hành được thay thế bởi thành phần tương ứng của

chương trình mới.

Cách dùng thông dụng nhất của execve() là ở trong tiến trình con được sinh bởi một **fork ()**, mặc dù nó cũng thỉnh thoảng được sử dụng trong các ứng dụng mà không có fork() .

#include <unistd.h>

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

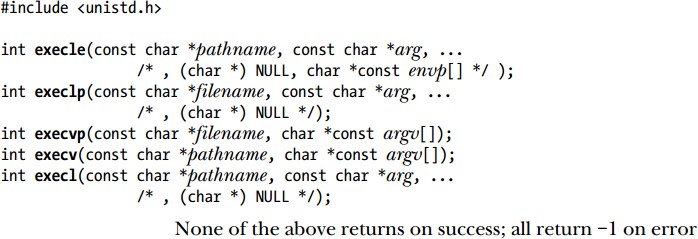
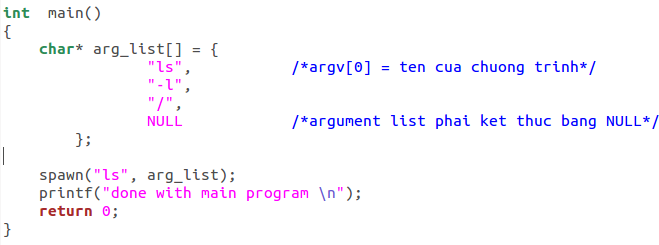
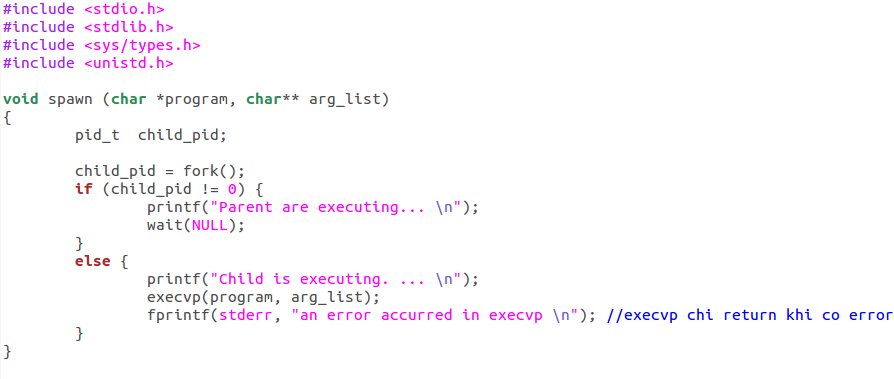
Never returns on success; returns –1 on error

* **pathname**: tên đường dẫn của chương trình mới. Đường dẫn này có thể là tuyệt đối hoặc tương đối - liên quan đến thư mục làm việc hiện tại của process gọi.

**argv :** chỉ định các đối số dòng lệnh truyền cho chương trình mới. Là một mảng con trỏ kết thúc bằng NULL, trỏ tới chuỗi ký tự. Giá trị cung cấp cho argv [0] tương ứng với tên lệnh.

**envp**: chỉ định danh sách môi trường cho chương trình mới. envp argument tương ứng với environ array của chương trình mới; nó là một danh sách con trỏ NULL-teminated dạng chuỗi : có cấu trúc name = value .

* Có nhiều hàm exec, là các API của system call execve(). Mỗi hàm có một giao diện khác nhau, nhưng chức năng tương tự :



*Tham khảo : man execve , man exec.*

**Chương trình fork03.c : thực thi một chương trình. Đọc hiểu chương trình , thực thi và xem kết quả ?**

**Bài tập 5 : Kết thúc một tiến trình và trạng thái kết thúc - exit() và wait()**

Hàm thư viện **exit(status)** kết thúc một tiến trình, tất cả các tài nguyên (memory, open file descriptors, …) đang được sử dụng bởi tiến trình sẵn sàng cho kernel phân bổ lại cho các tiến trình khác. Tham số **status** là một số nguyên xác định trạng thái kết thúc của tiến trình. Sử dụng lời gọi hệ thống **wait()** để có thể nhận tham số status này. (Tham khảo thêm \_exit() system call )

Lời gọi hệ thống **wait(&status)** có hai mục đích : thứ nhất, tiến trình cha tạm dừng cho đến khi một trong các tiến trình con của nó kết thúc (bằng exit(), … ). Thứ hai, trạng thái kết thúc của tiến trình con được trả về trong tham số **status** của wait().

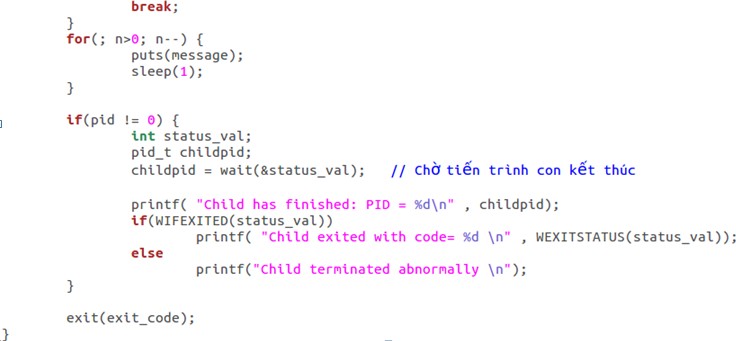
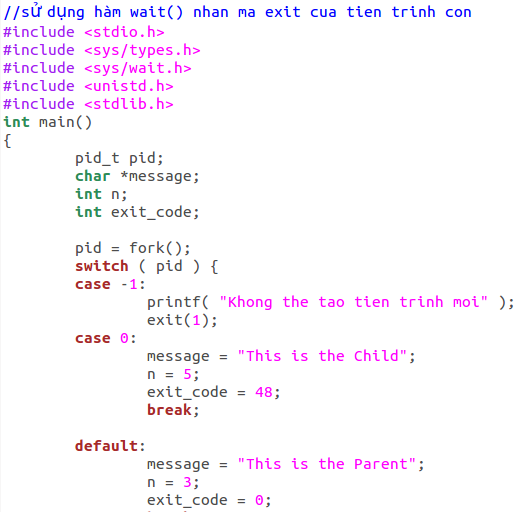
Có thể thông dịch thông tin về trạng thái kết thúc của tiến trình con sử dụng các macro định nghĩa trong **sys/wait.h :**

## WIFEXITED(stat\_val) Nonzero if the child is terminated normally. WEXITSTATUS(stat\_val) If WIFEXITED is nonzero, this returns child exit code. WIFSIGNALED(stat\_val) Nonzero if the child is terminated on an uncaught signal. WTERMSIG(stat\_val) If WIFSIGNALED is nonzero, this returns a signal number. WIFSTOPPED(stat\_val) Nonzero if the child has stopped.

**WSTOPSIG(stat\_val) If WIFSTOPPED is nonzero, this returns a signal number.**

*Tham khảo : man exit , man wait, man waitpid*

**Chương trình fork04.c : nhận trạng thái trả về của tiến trình con. Đọc hiểu chương trình, thực thi và xem kết quả ?**



# Bài tập ứng dụng :

1. **Zombie process**

Khi một tiến trình con kết thúc, thông tin về nó vẫn còn tồn tại trong hệ thống (trong process table) đến khi tiến trình cha kết thúc hoặc gọi hàm wait. Tiến trình con lúc này được gọi là zombie process ( hay defunct ).

Viết chương trình minh họa sự tồn tại của một Zombie process ?

1. **Độ ưu tiên của tiến trình :** getpriority() và setpriority() system call cho phép một tiến trình xem thông tin hay thay đổi nice value của chính nó hoặc của tiến trình khác . Tìm hiểu cú pháp, và viết một chương trình minh họa cách dùng 2 system call này ?

## #include <sys/resource.h>

**int getpriority(int which, id\_t who);** *//Returns (possibly negative) nice value of specified processon success, or –1 on error*

**int setpriority(int which, id\_t who, int prio);** *//Returns 0 on success, or –1 on error*