**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

**TÀI LIỆU:**

**HƯỚNG DẪN THỰC HÀNH**

**HỆ ĐIỀU HÀNH**

**Nhóm biên soạn:**

- ThS. Phan Đình Duy

- ThS. Phạm Văn Phước

- ThS. Nguyễn Việt Quốc

- KS. Nguyễn Hữu Nhân

- KS. Lê Văn La

- KS. Trần Văn Quang

***Tháng 3 năm 2015***

**NỘI DUNG CÁC BÀI THỰC HÀNH**

**Phần 1: Lập trình trên Linux**

Bài 1: Hướng dẫn cài đặt Ubuntu và các lệnh cơ bản của shell

Bài 2: Cơ bản lập trình shell

**Phần 2: Thực hành hệ điều hành**

**Bài 3: Quản lý tiến trình**

Bài 4: Định thời CPU

Bài 5: Đồng bộ hóa tiến trình, tiểu trình

Bài 6: Quản lý bộ nhớ

**Phần 3: Bài tập lớn**

CÁC PHẦN MỀM THIẾT BỊ SỬ DỤNG TRONG MÔN THỰC HÀNH

- Phần mềm VMware

- Hệ điều hành Ubuntu

# 

# Quản lý tiến trình, tiểu trình

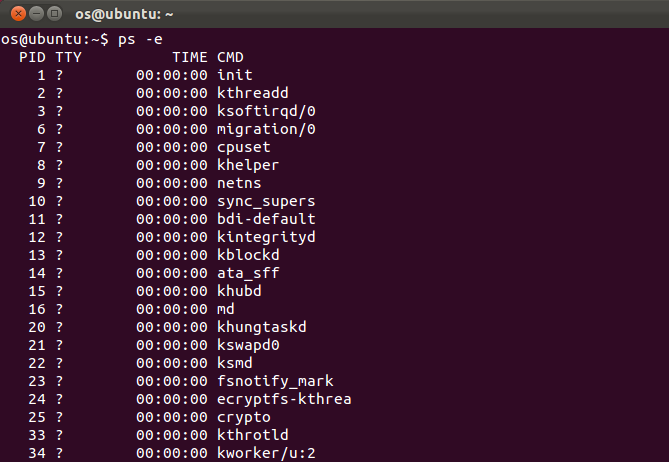
Sinh viên sẽ thực hành các thao tác liên quan tới tiến trình trong hệ điều hành. Mục đích của bài thực hành này là giới thiệu về tiến trình, tiểu trình trong hệ điều hành linux. Cuối cùng sinh viên sẽ áp dụng lập trình đa tiều trình (multithreading) nhằm giảm thời gian thực thi hiện song song. Bài thực hành bao gồm các phần chính:

* Tìm hiểu về tiến trình, tiểu trình.

## Tiến trình trong linux

Một tiến trình, trong linux được gọi là task, là một thực thể của một chương trình đang chạy trong linux. Chẳng hạn nếu 10 người trên một server cùng sử dụng một chương trình, như emacs (trình soạn thảo), thì ta có 10 tiến trình emacs cùng chạy trên server đó, mặc dù chúng đều xuất phát từ một chương trình.

Trong linux, ta có thể dùng lệnh ps –e để hiển thị các tất cả các tiến trình đang chạy trong hệ thống, hình minh họa

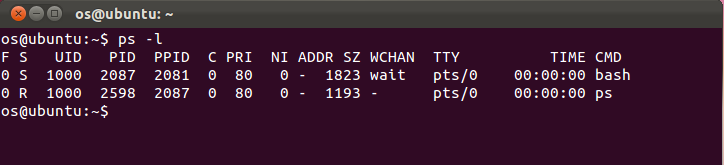


Hình 1. Tiến trình trong linux

Trong đó, PID là số id của tiến trình, TTY cho biết tiến trình thuộc vào terminal nào và CMD là tên của chương trình đang được chạy.

ID của tiến trình (PID) là duy nhất cho một tiến trình trong hệ thống. Hệ điều hành sử dụng biến counter 32-bit để đặt ID cho tiến trình. Một khi tiến trình được khởi tạo, biến counter sẽ được tang lên và giá trị của nó trở thành PID của tiến trình vừa được khởi tạo đó.

Để hiển thị nhiều thông tin các tiến trình gắn liền với một terminal ta gõ lệnh ps -l.



Hình 2. Hiển thị thông tin tiến trình gắn liền với terminal hiện tại

Cột S là một trong các cột quan trọng nhất, nó cho biết trạng thái của tiến trình. Các tiến trình có thể thuộc vào các trạng thái như bảng sau

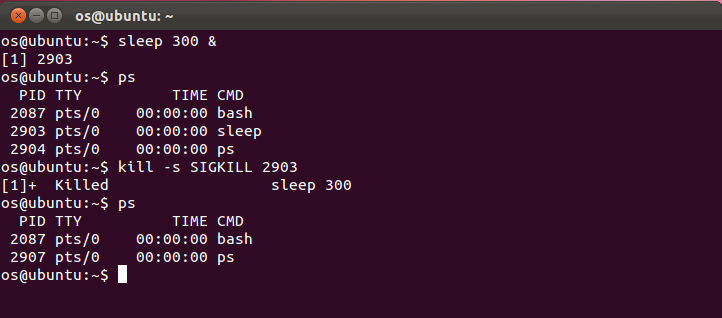
|  |  |
| --- | --- |
| Code | Name |
| D  R  S  T  Z | Uninteruptible sleep  Runnable (on run queue)  Sleeping  Traced hoặc Stopped  Defunct (zombie) |

Nếu ta sử dụng lệnh ps –el, ta nhận thấy rằng hầu hết các tiến trình đều đang trong trạng thái ngủ (sleeping – S). Bởi vì, tất cả đang đợi input để xử lý. Trong Hình 2, ta có thể thấy PPID của ps chính là PID của bash, do ps được kích hoạt từ bash, khi đó bash được gọi là tiến trình cha của tiến trình ps. Như vậy sau khi khởi tạo tiến trình ps xong, bash đã chuyển sang trạng thái ngủ.

Bởi vì có mối quan hệ cha-con giữa các tiến trình, nên linux hỗ trợ lệnh để hiển thị mối quan hệ này dưới dạng cây thông qua lệnh pstree. Ta có thể nhận ra rằng cây quan hệ này bắt nguồn từ init, sau đó đến các tiến trình cha (PPID) và các daemon (các tiến trình chạy ngầm không tương tác trực tiếp với người dùng).

## Signals

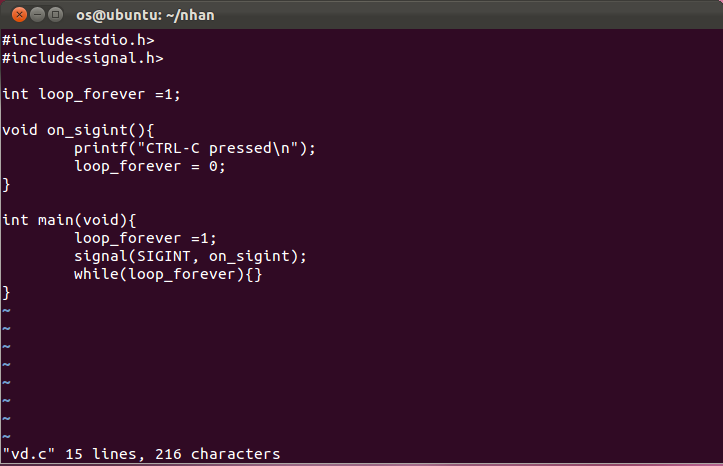
Trong linux, các tín hiệu được xem như một kênh giao tiếp bất đồng bộ với một tiến trình. Một signal ngắt một tiến trình đang chạy bình thường để chuyển hệ thống sang phục vụ sự kiện khác. Có rất nhiều loại signal trong linux, sinh viên có thể tham khảo trong tài liệu đính kèm. Từ terminal, ta có thể dùng kill gửi signal tới một tiến trình. Ví dụ:



Hình 3. Hủy tiến trình bằng lệnh trong terminal

Ví dụ trên dùng lệnh KILL để gửi tín hiệu SIGKILL tới tiến trình ngủ 300 giây. Ta có thể dùng lệnh này này để loại bỏ các tiến trình không mong muốn với các lựa chọn SIGKILL hoặc SIGTERM. Đối với SIGKILL tiến trình sẽ bị hủy ngay lập tức, trong khi SIGTERM thì tiến trình có thời gian để xử lý các thao tác.

Có 3 cách để bắt một tín hiệu trong lập trình. Nó có thể bị từ chối (không phải tất cả tín hiệu đều bị từ chối), nó có thể được gửi tới một handler mặc định hoặc gửi tới một handler định trước. Để có thể bắt tín hiệu định trước, ta có thể dùng hàm signal (trong thư viện signal.h) để lấy số hiệu và địa chỉ của một hàm để bắt tín hiệu. Ví dụ dưới đây bắt sự kiện CTRL-C (SIGINT – signal interrupt).



Hình 4. Bắt tín hiệu SIGINT

## Tạo tiến trình

### Tạo tiến trình từ tiến trình cha

Trong linux, tiến trình được tạo ra bằng cách nhân đôi một tiến trình bằn một lời gọi hệ thống fork. Trong C, ta có thể thực hiện bằng lệnh fork():

#include <sys/types.h>

#include <unistd.h>

pid\_t fork(void);

Sau khi tạo tiến trình con từ tiến trình cha, các tiến trình sử dụng chung code nhưng sử dụng stack và heap riêng. Hàm fork trả về giá trị là PID, ta có thể sử dụng giá trị này để phân biệt các process. Đối với tiến trình con, hàm fork trả về giá trị 0, nhưng trong hàm fork cha sẽ trả về PID của hàm mới tạo (nếu giá trị trả về âm là lỗi). Sinh viên chạy ví dụ sau:

#include <unistd.h> /\* Symbolic Constants \*/

#include <sys/types.h> /\* Primitive System Data Types \*/

#include <stdio.h> /\* Input/Output \*/

#include <stdlib.h> /\* General Utilities \*/

int main(){

pid\_t pid;

pid=fork();

if(pid==0)

printf("Child process, gia tri pid=%d\n",pid);

else

printf("Parent Proces, gia tri pid=%d\n",pid);

exit(0);

}

Hàm wait có thể được sử dụng để hàm cha đợi cho đến khi hàm con kết thúc. Ví dụ sau đây khiến tiến trình cha phải đợi tiến trình con hoàn thành trước.

#include <unistd.h> /\* Symbolic Constants \*/

#include <sys/types.h> /\* Primitive System Data Types \*/

#include <stdio.h> /\* Input/Output \*/

#include <stdlib.h> /\* General Utilities \*/

int main(){

pid\_t pid;

pid=fork();

if(pid==0)

printf("Child process, gia tri pid=%d\n",pid);

else {

wait(NULL);

printf("Parent Proces, gia tri pid=%d\n",pid);

}

exit(0);

}

### Khởi tạo tiến trình từ chương trình có sẵn

Hàm fork giúp tạo một tiến trình mới từ tiến trình hiện tại, tuy nhiên, nếu chúng ta muốn chạy một chương trình đã có sẵn thì chúng ta cần thực hiện một lời gọi hệ thống. Hàm execl sẽ thay thế tiến trình hiện tại bởi tiến trình mới được gọi từ một chương trình. Ví dụ:

#include<stdlib.h>

#include<unistd.h>

int main(){

execl("/usr/bin/gedit", "gedit", "foo.c”, NULL);

exit(1);

}

Nếu chúng ta muốn mở tạo một file với tên là foo.c, đồng nghĩa với việc ta gõ lệnh gedit foo.c trong terminal, ta có thể truyền các tham số cho hàm execl như execl("/usr/bin/gedit", "gedit", "foo.c”, NULL);

Trong lập trình ta có thể kết hợp lệnh fork và lệnh execl để tiến trình vừa tiếp tục chạy và vừa tạo thêm tiến trình mới từ việc mở thêm chương trình khác. Đầu tiên, ta dùng lệnh fork để tạo một tiến trình con mới. Sau đó, trong tiến trình con ta dùng hàm execl để khởi tạo tiến trình mới.

## Tiểu trình

Ở phần này, sinh viên sẽ làm quen với các thao tác cơ bản của tiểu trình. Tiểu trình trong linux được sử dụng thông qua gói thư viện chuẩn pthreads. Chúng ta cần thêm cờ -pthread khi biên dịch chương trình để có thể biên dịch chương trình có sử dụng tiểu trình. Vì dụ, nếu ta muốn biên dịch file có tên là threadprog.c ta sử dụng lệnh như sau:

$gcc –pthread –o threadprog threadprog.c

Ngoài ra, khi lập trình chúng ta cần khai báo thư viện pthreads vào bằng cách thêm pthread.h vào header.

### Khởi tạo tiểu trình

Một tiểu trình được tạo ra bằng cách gọi hàm pthread\_create. Hàm pthread\_create được định nghĩa đầy đủ như sau:

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arg);

* Pthread\_create trả về một số integer, nếu tiểu trình tạo thành công sẽ trả về 0, ngược lại là số khác 0.
* Đối số đầu tiên trỏ tới một kiểu cấu trúc có tên pthread\_t. Cấu trúc này giữ các thông tin hữu ích về một tiểu trình (thread). Vì thế, mỗi khi tạo mới một tiểu trình, ta cần cấp phát bộ nhớ để lưu trữ thông tin này.
* Đối số thứ hai là một con trỏ tới cấu trúc pthread\_attr\_t. Cấu trúc này chứa các thuộc tính của tiểu trình như số lượng bộ nhớ cần cấp phát cho stack của tiểu trình. Mặc định bộ nhớ cần thiết là 512K. Nếu số lượng tiểu trình quá nhiều có thể gây ra tràn bộ nhớ. Vì thế ta có thể cấp phát bộ nhớ cho stack nhỏ hơn.

#define SMALL\_STACK 131072 //128K for stack

pthread\_attr\_t thread\_attr;

pthread\_attr\_init(&thread\_attr);

pthread\_attr\_setstacksize(&thread\_attr, SMALL\_STACK);

Lưu ý, ta có thể sử dụng lại cấu trúc thuộc tính này cho các tiểu trình khác nhau.

* Đối số thứ 3 là con trỏ hàm chỉ tới hàm mà tiểu trình sẽ thực thi.
* Đối số cuối cùng là đối số được truyền cho hàm sẽ thực thi được khai báo ở đối số thứ 3.

### Ví dụ về tiều trình

Dưới đây là một ví dụ về sử dụng pthread\_create. Các dòng lệnh dưới sẽ tạo một tiểu trình thực thi hàm fn

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#define SMALL\_STACK 131072 //128K for stack

pthread\_attr\_t thread\_attr;

void\* fn(void\* arg);

int main(int argc, char\*\* argv)

{

pthread\_attr\_init(&thread\_attr);

pthread\_attr\_setstacksize(&thread\_attr, SMALL\_STACK);

pthread\_t th;

pthread\_create(&th, &thread\_attr, fn, (void\*)14);

void\* val;

thread\_join(th, &val);

return 0;

}

void\* fn(void\* arg){

printf("arg = 0x%x\n", (int)arg);

return NULL;

}

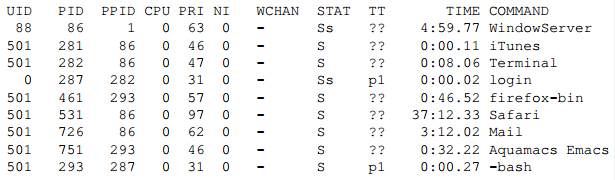
Hàm pthread\_join làm cho hàm main đợi cho đến khi tiểu trình kết thúc.

## Tổng kết

Bài thực hành này đã hướng dẫn sinh viên tìm hiểu về tiến trình trong linux. Qua đó, giúp sinh viên nắm được cách thức cơ bản trong lập trình như khởi tạo tiến trình, tiểu trình, bắt sự kiện trong lập trình,...

## Bài tập

1. Vẽ cây mối quan hệ của danh sách các tiến trình trong hình bên dưới. Tại mỗi nhánh, ghi chú tên và PID của tiến trình (nhớ rằng init luôn luôn có PID là 1)



1. Sinh viên viết một chương trình có chứa các handler của các sự kiện SIGHUP, SIGTERM, SIGINT. Đối với mỗi handler in ra dòng chữ cho biết chương trình đã bắt được tín hiệu thành công.

*Hướng dẫn: sau khi viết xong chương trình có tên ex-signals*

*Chạy lệnh* ./ex-signals

*Sau đó thực hiện gửi signal tới tiến trình đang chạy. VD:*

kill -SIGHUP pid

1. Cho chương trình sau:

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

int main(){

pid\_t pid;

int num\_coconuts = 17;

pid = fork();

if(pid == 0) {

num\_coconuts = 42;

exit(0);

} else {

wait(NULL); /\*wait until the child terminates \*/ }

}

printf("I see %d coconuts!\n", num\_coconuts);

exit(0);

}

Chương trình sẽ in gì ra màn hình? Giải thích tại sao.

1. Viết chương trình làm các công việc sau theo thứ tự:
   1. In ra dòng chữ “Thuc hanh he dieu hanh”
   2. Mở chương trình gedit
   3. Đợi người dùng nhấn CTRL-C để tắt.
   4. Khi người dùng nhấn CTRL-C, in dòng chữ “Ban da nhan CTRL-C”.
2. Sinh viên bỏ hàm pthread\_join trong ví dụ mục 1.4.2. Sau đó biên dịch và chạy lại chương trình. Chương trình có in ra màn hình giống với trước đó không? Giải thích tại sao.
3. Sinh viên tìm hiểu thêm về pthread\_create nhằm biết cách truyền nhiều đối số cho hàm thực thi của tiểu trình. Hãy hiện thực chương trình truyền MSSV và Họ Tên của mình cho tiểu trình và in ra màn hình.

PHỤ LỤC

[Chương 1. Quản lý tiến trình, tiểu trình 3](#_Toc412978816)

[1.1 Tiến trình trong linux 3](#_Toc412978817)

[1.2 Signals 5](#_Toc412978818)

[1.3 Tạo tiến trình 6](#_Toc412978819)

[1.3.1 Tạo tiến trình từ tiến trình cha 6](#_Toc412978820)

[1.3.2 Khởi tạo tiến trình từ chương trình có sẵn 8](#_Toc412978821)

[1.4 Tiểu trình 8](#_Toc412978822)

[1.4.1 Khởi tạo tiểu trình 9](#_Toc412978823)

[1.4.2 Ví dụ về tiều trình 10](#_Toc412978824)

[1.5 Tổng kết 10](#_Toc412978825)

[1.6 Bài tập 10](#_Toc412978826)

**TÀI LIỆU THAM KHẢO**

1. <http://www.ucs.cam.ac.uk/docs/course-notes/unix-courses/Building/files/signals.pdf>
2. <http://manpages.ubuntu.com/manpages/lucid/man2/fork.2.html>
3. <http://manpages.ubuntu.com/manpages/saucy/man3/pthread_create.3.html>
4. <http://manpages.ubuntu.com/manpages/saucy/man3/pthread_join.3.html>