**Processes and Threads**

-Một thread là một phần thực thi trong một process, một process có thể chứa nhiều thread.

-một thread có thể làm bất cứ thứ gì mà một process có thể làm, một thread có thể xem như là một process nhỏ, nhẹ.

-Threads được sứ dụng cho các nhiệm vụ nhỏ. trong khi đó process được sử dụng cho một nhiệm vụ lớn.

-các threads trong cùng 1 process thì có thể chia sẽ tài nguyên (địa chỉ bộ nhớ) , còn các process khác nhau thì không. Điều đó cho phé các threads có thể đọc và ghi các cấu trúc dữ liệu, các biến giống nhau và cũng có thể liên lạc vs nhau dễ dàng.

\***Thread:**

1.Thread id:

* process ID là duy nhất trên toàn hệ thống, trong đó thread ID là duy nhất trong một tiến trình (process).
* process ID là một giá trị số nguyên nhưng thread ID không nhất thiết phải là một giá trị số nguyên. Nó có thể là một **structure**.
* process ID có thể được in ra rất dễ dàng trong khi thread ID thì không.

Thread ID sẽ được đại diện bởi kiểu **pthread\_t**.

So sánh 2 thread dùng pthread\_equal(), một thread có thể thu được id của chính nó thông qua pthread\_self().

2.Tạo thread mới:

- Tạo thread mới pthread\_create()

Hàm pthread\_create() bao gồm 4 tham số, chúng ta sẽ cùng tìm hiểu về chúng.

* **Đối số đầu tiên** : Một khi tiến trình được gọi thành công, đối số đầu tiên sẽ giữ thread ID của thread mới được tạo.
* **Đối số thứ hai** : Thông thường giá trị này đặt thành NULL.
* **Đối số thứ ba** : Là một con trỏ hàm (**function pointer**) . Mỗi một thread sẽ chạy riêng một function, địa chỉ của function này sẽ được truyền tại đối số thứ ba để linux biết được thread này bắt đầu chạy từ đâu.
* **Đối số thứ tư** : Đối sô arg được truyền vào có kiểu void, điều này cho phép ta truyền bất kì kiểu dữ liệu nào vào hàm xử lý của thread. Hoặc giá trị này có thể là NULL nếu ta không muốn truyền bất cứ đối số nào. Điều này sẽ được thể hiện rõ ràng hơn trong ví dụ dưới đây.

3.Kết thúc thread

Một thead đang thực thi có thể bị kết thúc bởi một trong số những cách sau:

* Hàm xử lý của thread thực hiện return.
* Hàm xử lý của thread thực hiện gọi **pthread\_exit()**.
* Thread bị hủy bỏ bởi hàm **pthread\_cancel()**.
* Bất kì threads nào gọi **exit()** hoặc main thread thực hiện **return**. Nếu điều này xảy ra thì tất cả các threads còn lại sẽ bị kết thúc ngay lập tức.

Hàm pthread\_exit() kết thúc thread đang gọi và chỉ định giá trị trả về, giá trị này có thể nhận được trong một thread khác bằng cách gọi **pthread\_join()**.

\*\***Process:**

1, Tạo tiến trình mới:

+System call fork():

Một process mới có thể được tạo ra bằng việc sử dụng system call fork(). Process thực hiện gọi fork() được gọi là tiến trình cha (**parent process**). Process mới được tạo ra gọi là tiến trình con (**child process**).

 Sau khi lời gọi hàm fork() thành công, nó sẽ tạo ra một process con gần như giống với process cha ban đầu. Hai process này chia sẻ với nhau text segment, nhưng chúng sẽ có một bản sao riêng biệt đối với các segments còn lại là data, heap và stack. Điều này có nghĩa là, khi bạn thay đổi dữ liệu trong process con sẽ không ảnh hưởng tới dữ liệu trong process cha.

Nếu một process mới không được tạo ra, hàm fork() trả về -1.

2, Chạy chương trình mới:

Execl()theo tên chương trình và các tham số môi trường truyền vào cho chương trình đó.

int execl(const char \*path, char \*const argv[]);

3,Kết thúc tiến trình

+Kết thúc tiến trình bình thường:

Một process có thể hoàn thành việc thực thi của nó một cách bình thường bằng cách gọi system call **\_exit()**.

Đối số status truyền vào cho hàm \_exit() định nghĩa trạng thái kết thúc (**terminal status**) của process, có giá trị từ 0 - 255. Trạng thái này sẽ được gửi tới process cha để thông báo rằng process con kết thúc thành công (**success**) hay thất bại (**failure**). Process cha sẽ sử dụng system call **wait()** để đọc trạng thái này.

Trên thực tế, chúng ta sẽ không sử dụng trực tiếp system call \_exit() mà thay vào đó sẽ sử dụng **exit()** của thư viện **stdlib.h**.

#include <stdlib.h>

+ Kết thúc bất thường:

Dùng kill command để kết thúc tiến trình một cách chủ động

TerminatPID().