# Thực hành NLHDH buổi 1

## Tạo tiến trình con với lệnh fork

Cú pháp hàm: pid= fork() tạo ra 1 tiến trình con mới

pid = 0 nếu đang ở tiến trình con

pid<>0 là giá trị pid của con, nếu đang ở tiến trình cha

Ví dụ

if ((pid = fork()) == 0)

print "hello from the child"

else

print "hello from the parent"

**Bài số 1**:Tính cả tiến trình cha, có bao nhiêu tiến trình được tạo ra dưới đây?

int main() {

fork();

fork();

fork();

return 0;

}

**Bài số 2**: Nội dung được in ra màn hình sẽ gồm những gì?

void main() {

int pid;

pid = fork();

if (pid==0) { // it's the child process

fork(); printf("medium\n");

fork();

fork(); printf("small\n");

}

else printf("big\n");

}

**Bài số 3**: Cho đoạn mã chương trình dưới đây. Dòng A sẽ in ra kết quả gì?

int value = 5;

int main() {

pid\_t pid;

pid = fork();

if (pid == 0){ /\* child process \*/

value += 15;

return 0;

}

else if (pid > 0){ /\* parent process \*/

wait(NULL); /\* wait for the child process to terminate \*/

printf("PARENT: value = %d",value); // dòng A

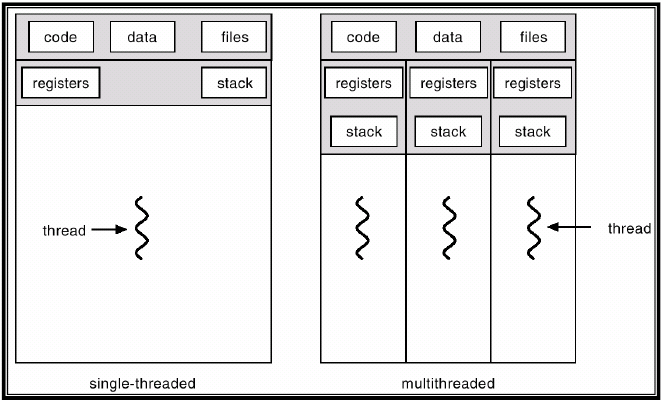
return 0;

}

}

## Lập trình với luồng

**Khái niệm về luồng**



Một tiến trình gồm có:

* đoạn mã chương trình (code);
* dữ liệu của tiến trình (data) gồm các biến tổng thể của chương trình trong phạm vi khai báo;
* các tài nguyên đang sử dụng (vd files đang mở);
* ngăn xếp (stack) chứa các tham số gọi hàm, các giá trị biến cục bộ khi gọi hàm/thủ tục con;
* các giá trị của tập thanh ghi (register) mà CPU đang thao tác trong đó có thanh ghi con trỏ lệnh (instruction pointer) chỉ ra vị trí của lệnh tiếp theo sẽ được CPU tải vào thực hiện.

Luồng là một môi trường con trong một tiến trình gồm có:

* Môi trường chung của các luồng trong tiến trình gồm đoạn mã chương trinh, dữ liệu, tệp đang mở
* Môi trường riêng:
  + các thanh ghi chứa các kết quả tính toán của CPU riêng đối với mỗi luồng gồm có con trỏ lệnh riêng, tức là mỗi luồng có chung 1 đoạn mã chương trình nhưng đang thực hiện tại các vị trí khác nhau
  + ngăn xếp cất giữ các tham số, biến cục bộ của hàm/thủ tục con

Với cơ chế đa luồng, một tiến trình có thể dùng cùng một mã chương trình để xử lý song song với các yêu cầu trên nhiều bộ dữ liệu khác nhau; các luồng chia sẻ chung dữ liệu tổng thể, tệp thao tác. Mỗi luồng vẫn xử lý công việc theo thuật toán tuần tự. Ví dụ một tiến trình trình web server có thể tạo 10 luồng để đáp ứng 10 yêu cầu duyệt web từ các trình duyệt; người lập trình không phải viết thuật toán xử lý song song để thực hiện công việc này.

**Các hàm thao tác với luồng**

|  |  |  |
| --- | --- | --- |
| Tạo 1 luồng | int pthread\_create (pthread\_t \*thread\_id, const pthread\_attr\_t \*attr, void\* (\*thread) (void \*), void \*arg); | thread\_id – số hiệu của luồng  attr – đặt thuộc tính luồng, NULL mặc định  thread – hàm được khởi chạy khi luồng tạo ra  arg – tham số cho hàm khởi tạo luồng |
| Chờ đồng bộ luồng con rồi mới kết thúc | int pthread\_join(pthread\_t thread\_id, void \*\*ret\_value); | thread\_id – số hiệu của luồng con  ret\_value – giá trị trả lại của luồng con qua pthread\_exit() |

**Cách biên dịch**

gcc –pthread –o <outputfile> <source.c>

**Ví dụ**:

#include <pthread.h>

#include <stdio.h>

char \*pHello="Hello world";

void \*myThread(void\* pStr) { // Đây là mã của luồng

printf("%s\n", (char\*) pStr);

}

void main() {

pthread\_t tid;

// tạo luồng \*myThread, truyền tham số pHello

pthread\_create(&tid, NULL, myThread, (void\*) pHello);

// cha chờ \*myThread kết thúc

pthread\_join(tid, NULL);

}

Dữ liệu chung của các luồng và tiến trình là biến tổng thể char\* pHello

Mã thực hiện luồng là hàm \*myThread, chạy song song với tiến trình cha trong main()

Tham số void\* pStr là dữ liệu thuộc phần không gian riêng của luồng \*myThread, được cất và lấy ra từ ngăn xếp.