# HỆ ĐIỀU HÀNH

# TUẦN 8

# TẠO TIỂU TRÌNH (LUỒNG - THREAD)

***Hướng dẫn làm bài***

**Luồng là gì ? Tại sao phải dùng luồng (thread)**

- Luồng là một phần của tiến trình sở hữu riêng ngăn xếp (stack) và thực thi độc lập ngay trong mã lệnh của tiến trình. Nếu như một HĐH có nhiều tiến trình thì bên trong mỗi tiến trình lại có thể tạo ra nhiều luồng hoạt động song song với nhau tương tự như cách tiến trình hoạt động song song bên trong HĐH.
- Ưu điểm của luồng là chúng hoạt động trong cùng không gian địa chỉ của tiến trình. Cơ chế liên lạc giữa các luồng đơn giản và hiệu quả.
- Đối với HĐH, chi phí chuyển đổi ngữ cảnh của tiến trình cao và chậm hơn chí phí chuyển đổi ngữ cảnh dành cho luồng.

## 1. Tạo threads

Thư viện hàm sử dụng:

`#include <pthread.h>`

Hàm khởi tạo thread

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
```

***Kết quả trả về***

▪ 0: Thành công, tạo ra một thread mới, ID của thread được trả về qua đối số thread

▪ <>0: thất bại

***Đối số truyền:***

▪ `thread`: dùng để giữ tham khảo đến threadID nếu hàm thành công. Kiểu pthread_t

▪ `attr`: giữ thuộc tính của thread, set **NULL** nếu muốn sử dụng các thuộc tính mặc định của hệ thống.

▪ `start_routine`: là địa chỉ của hàm do người sử dụng định nghĩa mà sẽ được thực thi bởi thread mới.

+ Hàm này nên có kiểu trả về là một con trỏ kiểu void, nếu không thì phải ép kiểu về con trỏ void khi gọi thread.

+ Hàm này nên có một đối số truyền cũng có kiểu con trỏ void.

+ Đối số truyền của hàm này sẽ được truyền thông qua tham số thứ 4 (`arg`)

▪ `arg`: là đối số truyền cho hàm `start_routine`.

Trong trường hợp đối số truyền cần ***nhiều hơn 1*** tham số có thể sử dụng **`struct`**, các đối số kiểu này phải được cấp phát tĩnh và ***được khởi tạo trước***.

Khi thread được tạo ra, nó có những thuộc tính riêng, một stack thực thi. Nó kế thừa các tín hiệu và độ ưu tiên từ chương trình gọi.

**Ví dụ 1:**

```
#include<pthread.h>
#include<stdio.h>

void * xinchao()
{
        printf("Hello. This is thread. \n");
}

int main()
{
        pthread_t thr;
        pthread_create(&thr,NULL,xinchao(),NULL);
}
```

**Kết quả:**

```
sv@SV-VirtualBox:~$ gcc -c thread.c
sv@SV-VirtualBox:~$ gcc -o thread.out thread.o -lpthread
sv@SV-VirtualBox:~$ ./thread.out
Hello. This is thread.
sv@SV-VirtualBox:~$
```

## 2. Đợi threads hoàn tất

Tương tự như tiến trình dùng hàm `wait()` để đợi tiến trình con kết thúc, bạn có thể gọi hàm `pthread_join()` để đợi một thread kết thúc.

```
int pthread_join(pthread_t th,
                 void ** thread_return);
```

▪ Kết quả trả về:

o Thành công: 0

o Thất bại: <>0

▪ Đối số:

- `th`: một threadID đã tồn tại
- `thread_return`: con trỏ đến vùng chứa giá trị trở về của luồng.

**Ví dụ 2:**

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int i;
void * xinchao(void* n)
{
        printf("This is thread %d\n",i);
        printf("Thread ID: %ld\n",(long int*) n);
        sleep(1);
}

int main(int argc, char* argv[])
{
        int num=atoi(argv[1]);
        pthread_t thr[num];
        for(i=0;i<num;i++)
        {
                pthread_create(&thr[i],NULL,xinchao,(void *) &thr[i]);
                pthread_join(thr[i],NULL);
        }
        return 0;
}
```

**Kết quả**

```
sv@sv-VirtualBox:~$ ./vd2.out 2
This is thread 0
Thread ID: 140724369380496
This is thread 1
Thread ID: 140724369380504
```

## 3. Kết thúc threads

Khi thread được tạo ra, nó thực thi hàm start_routine cho đến khi hoàn tất.

- Có lời gọi hàm thread_exit tường minh

```
void pthread_exit(void *retval);
```

- Bị cancel bởi hàm thread_cancel
- Tiến trình tạo ra thread kết thúc
- Có một thread gọi system call exec

**Ví dụ 3:**

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void* thr1(void* ar)
{
    int count;
    printf("this is thread %d\n",*((int*)ar));
    sleep(2);
}

int main (int argc, char* argv[])
{
    int i;
    pthread_t tid[3];
    int status, *pstatus = &status;
    for(i=0;i<3;i++)
        pthread_create(&tid[i],NULL,thr1,(void*) &tid[i]);
    for(i=0;i<3;i++)
    {
        if(pthread_join(tid[i],(void**) pstatus)>0)
        {
            printf("pthread_join for thread %d failure\n", (int)tid[i]);
        }
        printf("pthread_waited of %d OK, return code: %d\n", (int)tid[i], status);
        sleep(1);
    }
    sleep(1);
    return 0;
}
```

**Ví dụ 4:**

- Truyền dữ liệu cho thread

- Đối số thứ 4 là một kiểu struct

- *Lưu ý phần ép kiểu trong thr1.*

- *Lưu ý đối số truyền thứ 4 của hàm pthread_create*

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

struct arr
{
    int  n;
    int a[10];
};

void* thr1(void* ar)
{
    int count;
    struct arr *ap=(struct arr*) ar;
    for(count=0; count<ap->n; count++)
        printf("%d\t",ap->a[count]);
    printf("\n");
}

int main(int argc, char* argv[])
{
    struct arr ar;
    ar.n=5;
    int i;
    for(i=0;i<ar.n;i++)
        ar.a[i]=i+1;
    pthread_t tid;
    pthread_create(&tid,NULL,&thr1,&ar);
    sleep(2);
    return 0;
}
```

**Bài tập thực hành**

**Bài 1:** Viết chương trình đa luồng để xuất ra màn hình các số từ 1 đến n (n được truyền vào từ đối số của dòng lệnh). Luồng thứ nhất xuất ra các số lẻ. Luồng thứ 2 xuất ra các số chẵn.

**Bài 2:** Viết chương trình đa luồng tính toán các giá trị thống kê khác nhau từ một danh sách các số được truyền vào thông qua đối số của dòng lệnh. Chương trình sau đó sẽ tạo ba tiểu trình tính toán riêng biệt. Một tiểu trình sẽ xác định trung bình cộng của các số, tiểu trình thứ hai sẽ xác định giá trị lớn nhất và tiểu trình thứ ba sẽ xác định giá trị nhỏ nhất.

**Ví dụ:**

```
>./bai22.out 90 81 78 95 79 72 85
Gia tri trung binh: 82
Gia tri lon nhat: 95
Gia tri nho nhat: 72
```

Các biến số đại diện cho các giá trị trung bình, nhỏ nhất và lớn nhất sẽ được lưu trữ trên toàn cục. Các tiểu trình sẽ tính toán các giá trị này và tiến trình cha sẽ xuất ra các giá trị kết quả khi tiểu trình kết thúc.

**Bài 3:** Viết chương trình đa luồng để xuất ra số nguyên tố.

Người dùng chạy chương trình và nhập vào một số nguyên thông qua đối số tại dòng lệnh. Chương trình sau đó sẽ tạo ra một tiến trình riêng biệt xuất ra tất cả các số nguyên tố nhỏ hơn hoặc bằng số được nhập bởi người dùng.

--- HẾT ---