Multithreading (đa luồng)

Trước hết cần ta cần nhắc lại cách mà chương trình từ trước đến giờ chúng ta chạy:

- Chương trình sẽ vào hàm main thực hiện những câu lệnh từ trên xuống dưới.
- Nếu chúng ta sử dụng lời gọi một hàm nào đó, thì hàm main sẽ tạm thời ngừng lại, sao đó chương trình sẽ nhảy tới hàm được gọi để thực hiện từng câu lệnh (theo thứ tự từ trên xuống dưới)
  - ⇨ Nhận xét: như vậy với cách lập trình như trên , chúng ta thấy chương trình chỉ thực hiện trên 1 luồng (thread) duy nhất.

1. Multithreading (đa luồng) là gì?

- Khác với cách lập trình trên, MultiThreading (đa luồng) cho phép chương trình chạy cùng một lúc nhiều hàm hoặc xử lý cùng lúc nhiều công việc.
- Về cơ bản Multi Thread là một khả năng của một nền tảng (hệ điều hành, máy ảo …).
- Trong các hệ thống đa lõi và đa xử lý thì đa luồng tức là các thread được thực hiện cùng lúc trên lõi hoặc bộ vi xử lý khác nhau.
- Đối với hệ thống lõi đơn thì đa luồng chia thời gian giữa các thread. System sẽ gửi 1 số lượng nhất định các hướng dẫn từ mỗi Thread để xử lý. Các Thread không được thực hiện đồng thời. System chỉ mô phỏng thực hiện đồng thời của chúng. Tính năng này của System được gọi là đa luồng.
  - ⇨ Nhận xét: Multithreading được sử dụng khi thực hiện song song 1 số nhiệm vụ dẫn đến việc tận dụng hiệu quả hơn các tài nguyên của hệ thống.

2. Cách tạo một thread

Đầu tiên dĩ nhiên cần include thư viện Windows.h.

```
#include <Windows.h>
```

Sử dụng hàm **CreateThread** để tạo một thread (luồng)

```
HANDLE CreateThread(
       LPSECURITY_ATTRIBUTES lpThreadAttributes,
       SIZE_T dwStackSize,
       LPTHREAD_START_ROUTINE lpStartAddress,
       LPVOID lpParameter,
       DWORD dwCreationFlags,
       LPDWORD lpThreadId
    );
```

Hàm trả về một đối tượng có kiểu dữ liệu là HANDLE, đối tượng này được sử dụng để xử lý thread sau này.

Đối số truyền vào	**Ý nghĩa**
lpThreadAttributes	SECURITY_ATTRIBUTES dùng để miêu tả tính kế thừa, xem những thread có thể kế thừa hay không.
dwStackSize	Size vùng nhớ stack cấp cho thread
lpStartAddress	Hàm thực thi của thread
lpParameter dwCreationFlags	Đối số truyền vào hàm thực thi của thread Cờ dùng điều khiển một số mode của thread
lpThreadId	Con trỏ lưu trử ID của thread.

VD: chúng ta sẽ tạo ra một thread có tên là thread1 chạy song song với thread chính (hàm main)

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

WINAPI HAM_THUC_THI(LPVOID lpThreadParameter)
{
	while (1)
	{
		printf("[sub thread]: xin chao\r\n");
		Sleep(2000);
	}
}

void main()
{
	HANDLE thread1 = CreateThread(NULL, 16, HAM_THUC_THI, NULL, NULL, NULL);
	while (1)
	{
		printf("[main thread]: hello\r\n");
		Sleep(1000);
	}
}
```

3. Cách chấm dứt một thread

Sử dụng hàm **SuspendThread** để đình chỉ hoặc chấm dứt hoạt động của một thread:

```
DWORD SuspendThread(HANDLE hThread);
```

Đối số truyền vào là một đối tượng mà chúng ta muốn chấm dứt hoạt động của nó (có kiểu dữ liệu là HANDLE)

VD: Trong thread chính (hàm main) sẽ tăng dần biến cnt sao mỗi 1 giây, đến khi biến cnt lớn hơn 5 thì thread chính (hàm main) sẽ đình chỉ hoạt động của thread1.

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

WINAPI HAM_THUC_THI(LPVOID lpThreadParameter)
{
	while (1)
	{
		printf("[sub thread]: xin chao\r\n");
		Sleep(2000);
	}
}

void main()
{
	HANDLE thread1 = CreateThread(NULL, 128, HAM_THUC_THI, NULL, NULL, NULL);
	int cnt = 0;
	while (1)
	{
		printf("[main thread]: hello\r\n");
		Sleep(1000);
		if (cnt++ > 5)
		{
			SuspendThread(thread1);
		}
	}
```

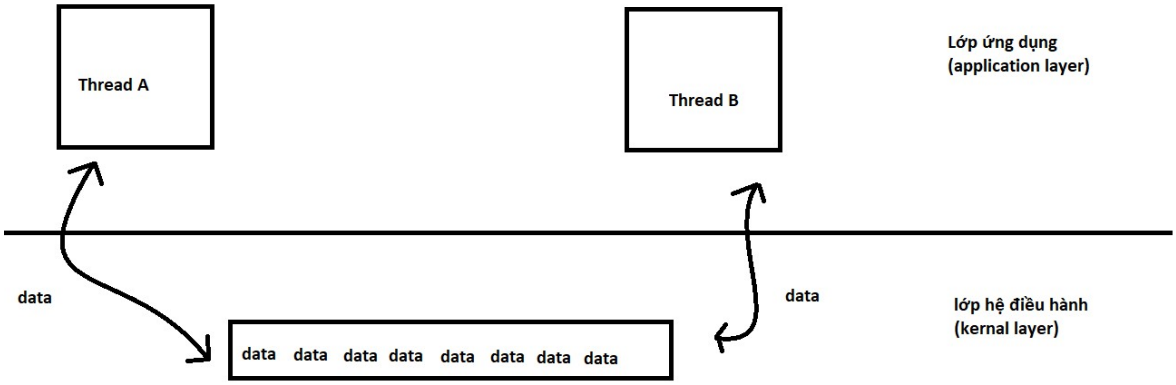
}

4. Cách truyền dữ liệu từ giữa các thread với nhau

4.1. Sử dụng biến toàn cục (cách đơn giản nhất)

4.2. Sử dụng pipe để truyền dữ liệu giữa các thread hoặc các proccess.

4.2.1. PIPE là gì.?



- **Pipe:** là một không gian vùng nhớ nằm ở kernal (lõi của hệ điều hành) nó cho phép các thread hoặc process truy cập vào.
- So sánh với vùng nhớ mà hệ điều hành cấp cho chúng ta khi chạy một chương trình ta thấy vùng nhớ mà hệ điều hành cấp, chúng ta chỉ được sử dụng trong chương trình đó, chương trình khác không có quyền truy cập đến vùng nhớ này . Nên pipe được tạo ra để các chương trình có trao đổi dữ liệu với nhau.

4.2.2. Cách sử dụng PIPE:

Tạo pipe: chúng ta sử dụng hàm CreatePipe để tạo 2 pipe (1 dùng để truyền dữ liệu, 2 dùng để nhận dữ liệu)

Cú pháp:

```
BOOL CreatePipe(
        _Out_ PHANDLE hReadPipe,
        _Out_ PHANDLE hWritePipe,
        _In_opt_ LPSECURITY_ATTRIBUTES lpPipeAttributes,
        _In_ DWORD nSize);
```

Đối số:

Đối số truyền vào	Ý nghĩa
hReadPipe	Địa chỉ của PIPE được tạo ra, dùng để đọc dữ liệu
hWritePipe	Địa chỉ của PIPE được tạo ra, dùng để truyền dữ liệu đi
lpPipeAttributes	Thuộc tính của PIPE
nSize	Kích thước của pipe (đơn vị byte)

Giá trị trả về:

- Nếu hàm được thực hiện thanh công, sẽ trả về một giá trị khác 0 (nonzero)
- Nếu hàm thực hiện không thành công (không thể tạo PIPE) hàm sẽ trả về giá trị 0.

Truyền dữ liệu vào PIPE: chúng ta sử dụng hàm WriteFile để truyền dữ liệu vào PIPE.

Cú pháp:

```
BOOL WriteFile(
    _In_ HANDLE hFile,
    _In_reads_bytes_opt_(nNumberOfBytesToWrite) LPCVOID lpBuffer,
    _In_ DWORD nNumberOfBytesToWrite,
    _Out_opt_ LPDWORD lpNumberOfBytesWritten,
    _Inout_opt_ LPOVERLAPPED lpOverlapped);
```

Đối số:

Đối số truyền vào	Ý nghĩa
hFile	Tên pipe truyền vào.
lpBuffer	Địa chỉ chứa dữ liệu muốn truyền đi
nNumberOfBytesToWrite	Số byte muốn truyền đi
lpNumberOfBytesWritten	Địa chỉ chứa số byte đã được truyền
lpOverlapped	Địa chỉ chứa OVERLAPPED struct. Có thể chứa giá trị NULL

Giá trị trả về:

- Nếu hàm thực hiện thành công, hàm sẽ trả về một giá trị khác 0 (nonzero)
- Nếu hàm thực hiện không thành công, hàm sẽ trả về một giá trị bằng 0.

*** ***Chú ý***: *để truyền giá trị đi, chúng ta sử dụng HANDLE* ***hWritePipe*** *(handle dùng để ghi dữ liệu)*

Đọc dữ liệu từ PIPE: chúng ta sử dụng hàm `ReadFile` để đọc dữ liệu từ PIPE

Cú pháp:

```
BOOL
ReadFile(
    _In_ HANDLE hFile,
    __out_data_source(FILE) LPVOID lpBuffer,
    _In_ DWORD nNumberOfBytesToRead,
    _Out_opt_ LPDWORD lpNumberOfBytesRead,
    _Inout_opt_ LPOVERLAPPED lpOverlapped);
```

Đối số:

Đối số truyền vào	Ý nghĩa
hFile	Tên pipe truyền vào.
lpBuffer	Địa chỉ chứa dữ liệu sao khi đọc (nơi chứa dữ liệu sao khi lấy từ pipe)
nNumberOfBytesToRead	Số byte muốn truyền đi
lpNumberOfBytesRead	Địa chỉ chứa số byte đã được truyền
lpOverlapped	Địa chỉ chứa OVERLAPPED struct. Có thể chứa giá trị NULL

Giá trị trả về:

- Nếu hàm thực hiện thành công, hàm sẽ trả về một giá trị khác 0 (nonzero)
- Nếu hàm thực hiện không thành công, hàm sẽ trả về một giá trị bằng 0.

*** ***Chú ý***: *để đọc giá trị về, chúng ta sử dụng HANDLE* ***hReadPipe*** *(handle dùng để đọc dữ liệu)*

VD:

```
#include <Windows.h>
#include <stdio.h>
HANDLE pipe2Read;
HANDLE pip2Write;

WINAPI HAM_THUC_THI(LPVOID lpThreadParameter)
{
	char data2Recv[2];
	while (1)
	{
		printf("[sub thread]: xin chao\r\n");
		int BytesRead = 0;
		if (ReadFile(pipe2Read, data2Recv, 1, &BytesRead, NULL))
		{
			printf("[sub thread] read OK\r\n");
			printf("[sub thread] data: %d\r\n", data2Recv[0]);
		}
		else
		{
			printf("[sub thread] write error: %d\r\n", GetLastError());
		}
```

```
		Sleep(1000);
	}
}

void main()
{
	char data2Send[] = {0,1,2,3,4,5,6,7,8,9};
	char index = 0;
	CreatePipe(&pipe2Read, &pip2Write, NULL, 16);
	HANDLE thread1 = CreateThread(NULL, 128, HAM_THUC_THI, NULL, NULL, NULL);
	while (1)
	{

		printf("[main thread]: hello\r\n");
		int BytesWritten = 0;
		if (WriteFile(pip2Write, &data2Send[index++], 1, &BytesWritten, NULL))
		{
			printf("[main thread] write OK\r\n");
		}
		else
		{
			printf("[main thread] write error: %d\r\n", GetLastError());
		}
		if (index >= 10)index = 0;
		Sleep(5000);
	}
}
```