Lập trình đa tiểu trình - Thread

1. Mục tiêu

SV tiếp cận bài toán đa tiểu trình và các thao tác trên tiểu trình.

2. Đặt vấn đề

Giả sử ta có n (n ≥2) công việc và muốn thực hiện n công việc này đồng thời. Có nhiều cách giải quyết vấn đề này, trong bài này sẽ giới thiệu một giải pháp là dử dụng thread, mỗi thread sẽ thực thi một công việc và thực thi song song với các công việc còn lại.

3. Tạo và hủy tiểu trình

1.1. Giới thiệu bài toán

Hãy viết chương trình theo kiểu bấm giờ bằng cách mỗi lần nhấn một phím thì sẽ in ra số nhịp đã trôi qua từ khi chương trình bắt đầu chạy, với mỗi nhịp bằng 100ms, chương trình sẽ kết thúc khi nhấn ESC. Với hàm sử dụng là hàm _getch, quyền điều khiển sẽ không được trả lại cho đến khi có một phím được nhấn, ta sẽ không thể thực hiện việc tăng biến đếm sau mỗi nhịp.

Có nhiều cách tiếp cận để giải quyết yêu cầu. Có thể dùng cơ chế Timer nhưng vẫn cần phải có một luồng (tiểu trình) khác để nhận WM_TIMER. Hoặc có thể kiểm tra chỉ khi nào phím đã được nhấn mới gọi hàm _getch, cách này khá phức tạp và không hiệu quả khi yêu cầu bài toán không chỉ là đếm giờ mà còn phải đọc đĩa, tương tác với mạng...

1.2. Tiếp cận bài toán theo hướng đa luồng

Giả sử ta có 2 luồng thực hiện 2 công việc riêng biệt (giống như 2 người làm 2 nhiệm vụ riêng biệt) như sau:

+ Luồng *CounterThread*: thực hiện tăng biến đếm nCount, delay trong 100ms, rồi lại tăng biến đếm nCount...

```
while (!bESCPressed)
{
        nCount++;
        Sleep(100);
}
```

+ Luồng *GetKeyThread:* nhận một phím bằng hàm _getch, in giá trị biến đếm nCount, rồi lại nhận một phím _getch... cho đến khi phím ESC được nhấn.

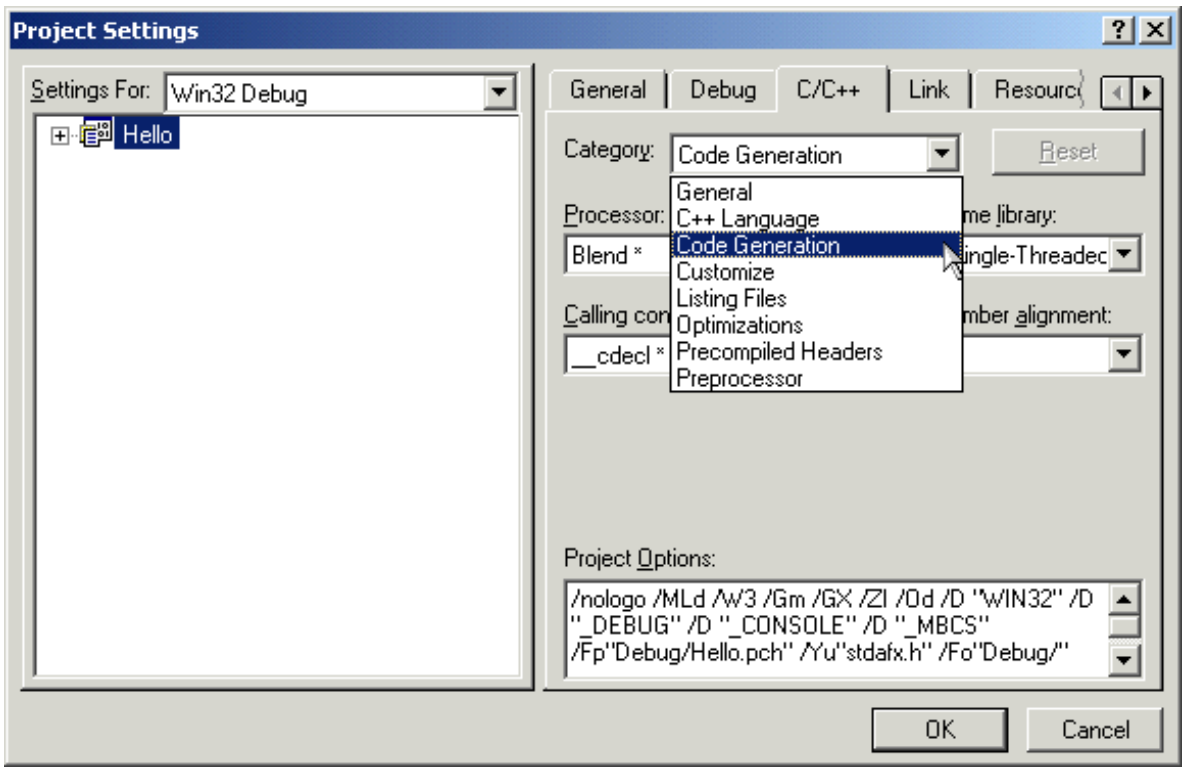
```
do {
        bESCPressed = (_getch() == 27);
        printf("%d\n", nCount);
```

```
} while (!bESCPressed);
```

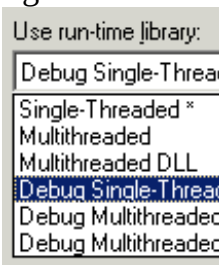
Rõ ràng với việc sử dụng 2 luồng cho bài toán trên, vấn đề trở nên đơn giản.

1.3. Điều kiện để một chương trình VC++ chạy được ở chế độ đa luồng

Nhấn Alt-F7 để vào trong Project\Settings...



Nếu như chế độ Use run-time library: đang để ở Single-Threaded hoặc Debug Single-Threaded thì phải chuyển sang chế độ Multithreaded tương ứng là Multithreaded và Debug Multithreaded.



chuyển thành ==>

Cách tạo một tiểu trình mới trong Visual C++

Trong Windows để khởi động một luồng mới, thì ta phải cài đặt luồng đó trong một hàm, khi hàm kết thúc thì luồng được tạo cũng tự kết thúc.

Để tạo một luồng mới ta dùng hàm **CreateThread**

```
HANDLE CreateThread
```

```
{
        SEC_ATTRS SecurityAttributes,
        ULONG StackSize,
        SEC_THREAD_START StartFunction,
        PVOID ThreadParameter,
        ULONG CreationFlags,
        PULONG ThreadId
};
```

Trong đó:

SecurityAttributes: trỏ đến cấu trúc SECURITY_ATTRIBUTES dùng để xác định handle của tiến trình mới tạo có được kế thừa bởi các tiến trình con không. Chú ý: trong Windows 9x nếu tham số này mang giá trị NULL thì handle này không thể kế thừa bởi tiến trình con khác, trong Windows NT nếu mang giá trị NULL có nghĩa là tiểu trình này sử dụng chế độ an toàn mặc định.

dwStackSize: kích thước ban đầu của stack cục bộ của tiểu trình.Nếu có giá trị 0 hay nhỏ hơn kích thước mặc định thì hệ thống dùng kích thước của tiểu trình tạo và tự động tăng kích thước khi cần. Nếu không đủ bộ nhớ thì hàm tạo tiểu trình mới sẽ thất bại. Vùng stack này tự động được giải phóng khi tiểu trình kết thúc.

lpStartAddress: tên hàm mô tả công việc mà tiểu trình cần thực hiện. Hàm này được khai báo như sau:

DWORD WINAPI ThreadProc(LPVOID *lpParameter*);

lpParameter: tham số được truyền cho tiểu trình. Lưu ý: tham số truyền vào phải ép kiểu thành LPVOID sau đó vô hàm ép kiểu ngược trở lại.

dwCreationFlags: qui định trạng thái tiểu trình khi mới tạo lập. Nếu mang giá trị 0, tiểu trình tạo ra sẽ thực thi ngay; nếu là CREATE_SUSPENDED , tiểu trình sẽ chờ cho đến khi gọi hàm ResumeThread.

lpThreadId: con trỏ nhận giá trị trả về là ID của tiểu trình.

Nếu thành công, nhận giá trị trả về là handle của tiểu trình. Ngược lại nhận giá trị NULL.

Mỗi tiểu trình sau khi được tạo đều có cùng độ ưu tiên mặc định là THREAD_PRIORITY_NORMAL. Ta có thể thay đổi hay xác định bằng hàm SetThreadPriority và GetThreadPriority.

hoặc sử dụng hàm:

```
CWinThread* AfxBeginThread(
        AFX_THREADPROC pfnThreadProc,
        LPVOID pParam,
        int nPriority = THREAD_PRIORITY_NORMAL,
        UINT nStackSize = 0,
        DWORD dwCreateFlags = 0,
```

```
        LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL );

CWinThread* AfxBeginThread(
        CRuntimeClass* pThreadClass,
        int nPriority = THREAD_PRIORITY_NORMAL,
        UINT nStackSize = 0,
        DWORD dwCreateFlags = 0,
        LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL );
```

Trong đó:

pfnThreadProc: tên hàm mô tả công việc mà tiểu trình cần thực hiện.

```
UINT MyControllingFunction( LPVOID pParam );
```

pParam: tham số được truyền cho tiểu trình.
nPriority: độ ưu tiên của thread.
pThreadClass: tên lớp kế thừa từ CWinThread.
Hàm này trả về con trỏ kiểu CWinThread nếu thành công, ngược lại trả về NULL

Ví dụ: khi muốn tạo tiểu trình CouterThread, ta thực hiện

Cách 1:

Hàm CounterThread được cài đặt như sau:

```
DWORD WINAPI CounterThread(LPVOID /*pParam*/)
{
        while (!bESCPressed) {
                nCounter++;
                Sleep(100);
        }
        return 0;
}
```

do *pParam* ít sử dụng nên ta không cần khai báo để tránh bị lỗi warning

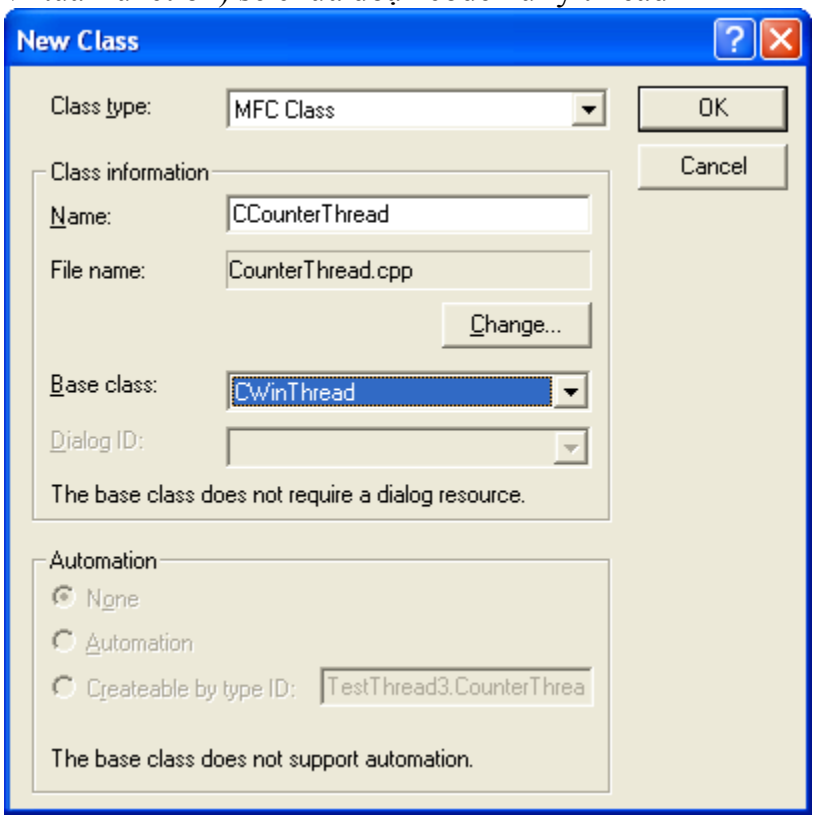
```
// tạo thread
DWORD dwThreadIdCounter;
HANDLE hThreadCounter;
hThreadCounter = CreateThread(NULL, 0,
        CounterThread, NULL, 0, &dwThreadIdCounter);
```

Cách 2:

```
// hàm xử lý thread cài đặt giống cách 1
// tạo thread
CWinThread *h1;

 h1 = AfxBeginThread((AFX_THREADPROC) CounterThread, NULL,
                        THREAD_PRIORITY_NORMAL,0,0,NULL);
```

Cách 3: tạo 1 lớp kế thừa từ CWinThread: CCounterThread, khi đó hàm Run (add virtual function) sẽ chứa đoạn code xử lý thread



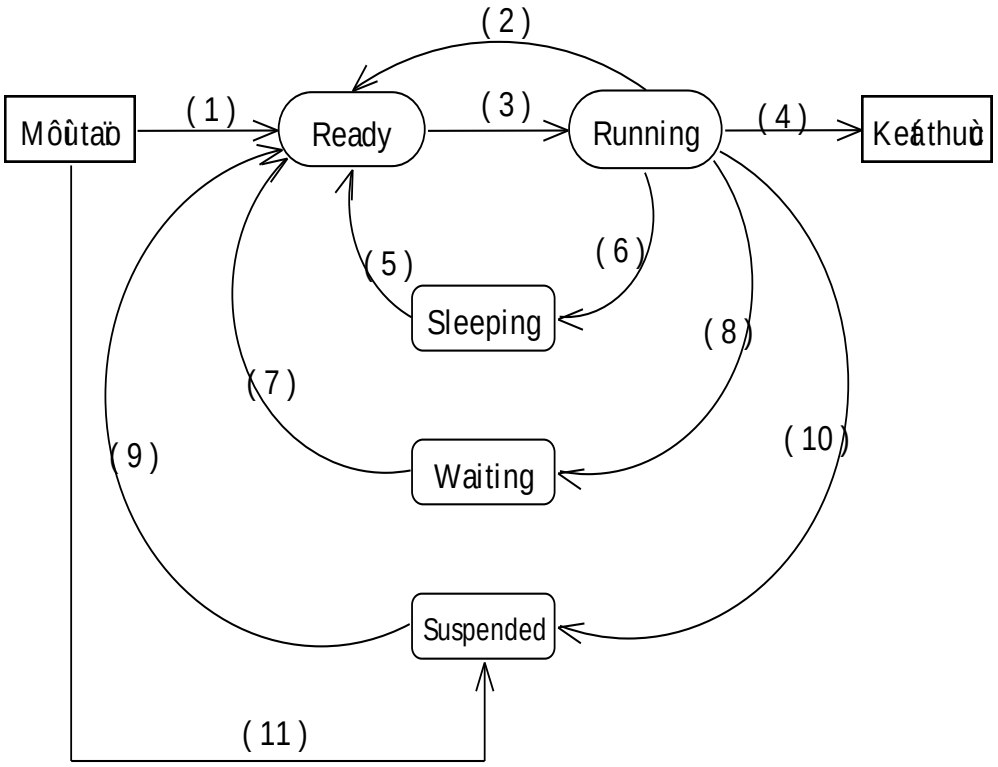

```
// Hàm Run của lớp CCounterThread được cài đặt như sau:
int CCounterThread::Run()
{
	// TODO: Add your specialized code here and/or call the base class
	while (!bESCPressed)
	{
		nCounter++;
		Sleep(100);
	}

	return CWinThread::Run();
}

// tạo thread
CCounterThread* h1;

h1 = (CCounterThread*)AfxBeginThread(RUNTIME_CLASS(CCounterThread),
                                     THREAD_PRIORITY_NORMAL,0,0,NULL);
```

1.4. Thay đổi trạng thái của tiểu trình



1.4.1. Hàm tạm dừng 1 tiểu trình

```
DWORD      SuspendThread
{
        HANDLE     hThread
};
```

Trong đó:

hThread: handle của tiểu trình muốn tạm dừng

Hàm SuspendThread tạm dừng hoạt động của 1 tiểu trình. Giá trị trả về số lần yêu cầu tạm dừng (suspend count) trước khi tăng hoặc 0xFFFFFFFF nếu có lỗi. (suspend count sẽ được tăng lên 1 mỗi lần yêu cầu tạm dừng tiểu trình).

1.4.2. Hàm Huỷ trạng thái tạm dừng của 1 tiểu trình

```
DWORD      ResumeThread
{
        HANDLE     hThread
};
```

Trong đó:

hThread: handle của tiểu trình muốn huỷ trạng thái tạm dừng

Hủy bỏ trạng thái tạm dừng của 1 tiểu trình bằng cách kiểm tra giá trị của số lần yêu cầu tạm dừng (suspend count), nếu suspend count > 0 , thì sẽ giảm đi 1 để huỷ bỏ 1 lần yêu cầu tạm dừng tiểu trình tương ứng, khi suspend count = 0, ResumeThread sẽ khôi phục lại hoạt động của tiểu trình. Giá trị trả về của hàm này là suspend count lúc chưa giảm hoặc 0xFFFFFFFF nếu có lỗi.

1.5. Kết thúc một tiểu trình

Ngoài việc một tiểu trình tự kết thúc khi hàm cài đặt của tiểu trình kết thúc, ta còn có thể yêu cầu tiểu trình kết thúc tức thời. Tuy nhiên, việc kết thúc hàm theo kiểu "thô bạo" như vậy sẽ dẫn đến việc không hủy bỏ đúng đắn những tài nguyên đã được cấp như mở file, kết nối mạng...

Kết thúc tiểu trình hiện hành: dùng hàm **ExitThread(dwExitCode)** trong đó dwExitCode là mã lỗi trả về, thường là bằng 0.

Để kết thúc một tiểu trình khác ta phải có handle của tiểu trình đó (hThread) được trả về khi tạo tiểu trình, khi đó dùng hàm **TerminateThread(hThread, dwExitCode)** để kết thúc một tiểu trình.

1.6. Các hàm khác

GetThreadTimes

```
BOOL        GetThreadTimes
{
        HANDLE          hThread,
        LPFILETIME      lpCreationTime,
        LPFILETIME      lpExitTime,
        LPFILETIME      lpKernelTime,
        LPFILETIME      lpUserTime
};
```

Trả về thời điểm tạo lập tiểu trình lpCreationTime, thời điểm kết thúc tiểu trình lpExitTime, thời gian tiểu trình hoạt động trong chế độ User lpUserTime, trong chế độ Kernel lpKernelTime của tiểu trình hThread.

Hàm GetExitCodeThread

```
BOOL        GetExitCodeThread
{
        HANDLE      hThread,
```

```
        LPDWORD  lpExitCode
};
```

Xác định trạng thái kết thúc của tiểu trình hThread. Nếu tiểu trình đang hoạt động, ExitCode nhận giá trị STILL_ACTIVE.

Hàm GetCurrendThreadId

```
HANDLE     GetCurrentThreadId (void);
```

Trả về định danh duy nhất của tiểu trình đang hoạt động.

Hàm GetCurrentThread

```
HANDLE     GetCurrentThread (void);
```

Trả về handle của tiểu trình đang hoạt động.

Lưu ý:

- Đối với lớp CWinThread, các hàm trên đều là hàm thuộc lớp cWinThread chỉ có hàm Terminate là không có. Tuy nhiên, có thể thao tác bình thường với các thread đó thong qua hThread trong lớp CWinThread.
- Trong môi trường C#, các bạn sẽ sử dụng lớp Thread trong System.Threading.Thread, sử dụng các hàm: Start, ThreadStart, Resume, Suspend, Abort…. (tham khảo thêm VD trong MSDN)

4. Bài tập

Bài tập 1: Viết chương trình tăng giảm số. Chương trình gồm có 2 tiểu trình, tiểu trình thứ nhất thực hiện việc tăng giá trị của biến x, trình thứ hai thực hiện việc giảm giá trị của biến x. Sau khi tăng giảm xong xuất nội dung của biến đó ra màn hình.

Bài tập 2: Viết chương trình chương trình hiển thị 10 ký tự ngẫu nhiên trên màn hình, mỗi ký tự được quản lý bởi một tiểu trình và di chuyển liên tục với tốc độ tùy ý. Khi di chuyển nếu chạm biên màn hình thì ký tự sẽ xuất hiện lại tại giữa màn hình. Chương trình kết thúc khi người dùng nhấn phím bất kỳ.

Bài tập 3: Viết chương trình tạo ra 3 tiểu trình, mỗi tiểu trình quản lý một ô vuông có màu khác nhau di chuyển tự do trên màn hình. Chương trình cho phép start, stop, suspend, resume, set priority từng tiểu trình.

Bài tập 4: Viết chương trình có 1 dấu * di chuyển chéo trên màn hình, khi gặp biên dấu * này sẽ dội lại (phản xạ). Tốc độ di chuyển giữa 2 ô kế nhau là 500ms. Khi người dùng nhấn các phím ↑↓←→ hoặc các phím W, Z, A, S thì ngay lập tức dấu * sẽ di chuyển lên trên, xuống dưới, sang trái, sang phải. Chương trình kết thúc khi người dùng nhấn phím ESC.

Bài tập 5: Viết chương trình cuộn chuỗi. Chương trình có hai tiến trình thực hiện thao tác sau: tiến trình thứ nhất thực hiện thao tác cuộn chuỗi, tiến trình thứ hai xuất chuỗi đang cuộn ra màn hình.

Lưu ý:

- Các bài tập không đặt nặng giao diện, chỉ yêu cầu hiểu được cách tạo thread và các thao tác trên thread.
- Đối với bài 1, 5: các bạn nên làm trên Console.
- Đối với các bài tập 2,3,4 nếu các bạn làm trên Console, các bạn xem thêm file hướng dẫn console.doc để biết cách xuất 1 ký tự ra màn hình tại 1 toạ độ xy. Nếu làm trên MFC, các bạn làm dạng View (Single Document)

Các câu hỏi thường gặp khi tạo thread:

1. ***Trong console, tạo thread nhưng thread không chạy.***

Trả lời: do hàm main sau khi tạo thread xong kết thúc luôn chương trình ➔ tất cả các tiểu trình đều kết thúc.

2. ***Truyền nhiều tham số vào trong 1 thread.***

Trả lời: nếu muốn truyền nhiều tham số vào trong 1 thread thì phải tạo 1 struct chứa các tham số đó sau đó truyền struct đó vào hàm.

```
MyStruct b;
b = *(MyStruct*)lp; // chỉ cần lấy nội dung của tham số truyền vào

AfxBeginThread((AFX_THREADPROC)CounterThread,(LPVOID)&b,
                THREAD_PRIORITY_NORMAL,0,0,NULL);
```

Nếu sử dụng lớp CWinThread, bạn nên add các biến thành viên cho lớp và gán trị trược tiếp. Khi đó, bạn cần tạo thread với cờ là CREATE_SUNSPENDED sau đó gọi hàm ResumeThread