# Process

## Tổng quan

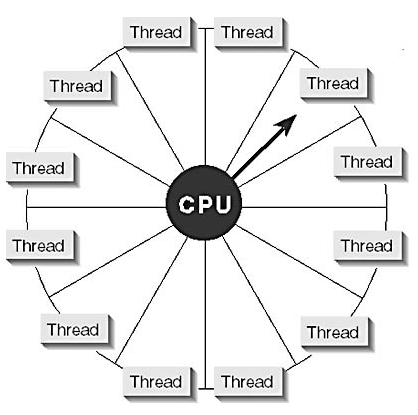
Một process thì thường được định nghĩa giống như một thể hiện của một chương trình đang chạy và bao gồm 2 thành phần

Một kernel project mà hệ thống dùng để quản lý process. Kernel object là nơi mà hệ thống lưu vết về các thông tin của process

Một vùng nhớ chứa mã nguồn và data của file thực thi hoặc DLL. Nó bao gồm các vùng nhớ stack và heap.

Process là nằm bị động. Để một process có thể hoàn thành một việc gì đó thì nó phải có ít nhất một thread chạy trong ngữ cảnh của nó. Thread này có nhiệm vụ thực thi mã nguồn chứa trong không gian vùng nhớ của process. Trong thực tế, một process có thể có nhiều thread và các thread này thực thi mã nguồn của process một cách đồng thời. Để làm được điều này thì mội thread phải có CPU Register và stack memory riêng. Mỗi process phải có ít nhất một thread thực thi các mã nguồn bên trong vùng nhớ của process. Nếu không có một thread nào thực thi mã nguồn của process thì không có lý do gì mà process đó tiếp tục tồn tại. Khi đó hệ thống sẽ hủy process cùng vùng nhớ của nó.

Để cho tất cả các thread được run thì OS phải lập lịch thời gian thực thi của CPU cho mỗi thread. Để tạo cảm giác như các thread thực thi đồng thời thì HĐH chia thời gian thực thi CPU thành time slice để thực thi các thread. Dưới đây là thuật toán lập lịch Round-Robin dùng cho hệ thống có 1 CPU



Vì thời gian cho một time slice rất ngắn (khoảng 20 ms). Do đó ta có cảm giác như các thread thực thi đồng thời.

Khi một process được tạo thì OS sẽ tự động tạo 1 thread cho nó. Thread này gọi là primary thread. Trong primary thread này có thể tạo thêm nhiều thread khác. Trong các thread khác đó lại có thể tạo thêm nhiều các thread khác nữa

## Create process

BOOL CreateProcess(

PCTSTR pszApplicationName,

PTSTR pszCommandLine,

PSECURITY\_ATTRIBUTES psaProcess,

PSECURITY\_ATTRIBUTES psaThread,

BOOL bInheritHandles,

DWORD fdwCreate,

PVOID pvEnvironment,

PCTSTR pszCurDir,

PSTARTUPINFO psiStartInfo,

PPROCESS\_INFORMATION ppiProcInfo);

Khi một thread gọi CreateProcess thì hệ thống sẽ tạo ra mối process kernel object **with an initial usage count of 1**. Đối tượng process kernel object này là một cấu trúc dữ liệu lưu các thông tin (số liệu) của process nằm giúp giúp OS quản lý process này. Hệ thống sau đó sẽ tạo vũng nhớ cho process và load mã nguồn và data của file thực thi hoặc DLL vào trong vùng nhớ này.

Tiếp theo hệ thống sẽ tạo một đối tượng thread kernel object để lưu trữ thông tin của primary thread của process vừa tạo. Primary thread bắt đầu bắt gọi **the C/C++ run-time startup code**, và trong mã nguồn startup này sẽ thực hiện hàm main. Nếu hệ thống tạo process và primary thread thành công thì hàm CreateProcess này trả về true

Chú ý

CreateProcess return trước khi process thật sự được khởi tạo đầy đủ. Nếu trong process mới tạo mà có thực hiện load DLL, mà các DLL này load thất bại thì process này sẽ bị hủy(nhưng hàm CreateProcess vẫn trả về true)

Tham số

pszApplicationName : đường dẫn **đầy đủ** đến file exe thực thi. 99% người ta pszApplicationName = NULL. Thay vào đó người ta dùng pszCommandLine để chỉ định process nào sẽ được thực thi.

pszCommandLine : sử dụng đường dẫn tương tự như trong command line để cho biết process nào sẽ được thực thi.

pszCommandLine có thể là đường dẫn tương đối và có thể không có phần mở rộng. HĐH sẽ giả sử phần mở rộng là EXE

vd : TCHAR szCommandLine[] = TEXT("mspaint");

Thứ tự tìm kiếm file thực thi như sau

Trong folder chứa file .exe của process gọi hàm CreateProcess

Trong foler System/System32

Trong folder Window

bInheritHandles

Giả sử một process cha (tên A) tạo ra một process con (tên B) thì process A có toàn quyền tác động vào process B. Trong trường hợp process A tạo một process mới (tên C), nếu ta muốn C cũng có các quyền A có thì bInheritHandles khi tạo C được chỉ định bằng true

psaProcess

Con trỏ tới cấu trúc SECURITY\_ATTRIBUTES. Trong cấu trúc này có trường bInheritHandle. Nếu trường này = TRUE thì các process con khác của process cha cũng có quyền truy cập vào process mới này.

psaThread

Con trỏ tới cấu trúc SECURITY\_ATTRIBUTES. Trong cấu trúc này có trường bInheritHandle. Nếu trường này bằng TRUE thì các process con khác của process cha cũng có quyền truy cập vào primary thread của process mới vừa tạo

fdwCreate

Cờ cho biết sẽ thực hiện tạo process mới theo cách thức nào

DEBUG\_PROCESS : cờ này cho hệ thống biết process cha muốn debug process con và con cháu của process con đó. Cờ này bảo hệ thống hãy thông báo cho process cha mỗi khi có một sự kiện nào đó xảy ra ở các process con cháu.

DEBUG\_ONLY\_THIS\_PROCESS : tương tự DEBUG\_PROCESS nhưng chỉ debug với process con trực tiếp. Không debug với các process con cháu của process con đó.

CREATE\_SUSPENDED : tạo process nhưng primary thread của process đó không thực thi ngay sau khi process được tạo. Điều nảy cho phép process cha chỉnh sửa vùng nhớ bên trong không gian vùng nhớ của process con, thay đổi độ ưu tiên của primary thread của process con và thêm process vào **a job** trước khi nó bắt đầu được thực thi. Ngay khi process cha đã chỉnh sửa xong process con thì có thể cho primary thread của process đó thực thi bằng cách gọi hàm ResumeThread.

DETACHED\_PROCESS : cờ này sử dụng khi một process dạng console tạo một process con cũng console, nhưng không muốn process con dùng chung cửa sổ với process cha. Khi đó một cửa sổ console mới sẽ được tao ra cho process con

CREATE\_NEW\_CONSOLE : tạo cửa sổ console mới cho process. Nếu DETACHED\_PROCESS và CREATE\_NEW\_CONSOLE thì tạo process thất bại

CREATE\_NO\_WINDOW : muốn hệ thống không tạo cửa sổ console cho process. Đây là một cờ hữu ích khi thực thi các ứng dụng console mà không cần giao tiếp gì với người dùng.

**CREATE\_NEW\_PROCESS\_GROUP**

CREATE\_DEFAULT\_ERROR\_MODE : process mới không kết thừa ErrorMode từ process cha. ErrorMode là thỏa thuận xử lý lỗi giữa hệ thống và process (được chỉ định bằng hàm SetErrorMode)

CREATE\_UNICODE\_ENVIRONMENT : process mới được tạo có thể sử dụng kiểu ký tự unicode. Mặc định thì process được tạo chỉ dùng ký tự ANSI

**CREATE\_BREAKAWAY\_FROM\_JOB**

fdwCreate còn cho phép chỉ định độ ưu tiên của tiến trình. Việc chỉ định độ ưu tiên nảy sẽ ảnh hưởng đến sự lập lịch các tiến trình của hệ thống. Sau đây là sắp xếp theo thứ tự tăng cần của độ ưu tiên

|  |  |
| --- | --- |
| Priority Class | Flag Identifier |
| Idle | IDLE\_PRIORITY\_CLASS |
| Below normal | BELOW\_NORMAL\_PRIORITY\_CLASS |
| Normal | NORMAL\_PRIORITY\_CLASS |
| Above normal | ABOVE\_NORMAL\_PRIORITY\_CLASS |
| High | HIGH\_PRIORITY\_CLASS |
| Realtime | REALTIME\_PRIORITY\_CLASS |

Mặc định hệ điều hành sẽ gán độ ưu tiên cho thread mới là Normal. Khuyến cáo : Nếu không rành về các thuật toán lập lịch của hệ thống thì nên để mặc định độ ưu tiên khi tạo process

**pvEnvironment**

**pszCurDir**

psiStartInfo

Con trỏ đến cấu trúc chỉ định thuộc tính của procss mới được tạo

ppiProcInfo

Con trỏ đến cấu trúc sẽ chứa thông tin của process mới vừa tạo

## OpenProcess

HANDLE WINAPI OpenProcess(

\_\_in DWORD dwDesiredAccess,

\_\_in BOOL bInheritHandle,

\_\_in DWORD dwProcessId

);

Hàm này lấy về HANDLE của một process khi biết ID của process đó

dwProcessId : id của process muốn lấy handle

dwDesiredAccess : cho biết muốn thực hiện quyền truy cập nào trên process được lấy handle

|  |  |
| --- | --- |
| DELETE |  |
| READ\_CONTROL |  |
| WRITE\_DAC |  |
| WRITE\_OWNER |  |
| PROCESS\_ALL\_ACCESS | Có mọi quyền truy cập vào process |
| PROCESS\_CREATE\_PROCESS |  |
| PROCESS\_CREATE\_THREAD | Tạo thread |
| PROCESS\_DUP\_HANDLE |  |
| PROCESS\_QUERY\_INFORMATION | Lấy các thông tin nào đó của process |
| PROCESS\_SET\_INFORMATION |  |
| PROCESS\_SET\_QUOTA |  |
| PROCESS\_VM\_OPERATION |  |
| PROCESS\_VM\_READ |  |
| PROCESS\_VM\_WRITE |  |

## Exit Process

Một process bị hủy khi một trong các thread của process đó gọi hàm ExitProcess

VOID ExitProcess(UINT fuExitCode);

Khi gọi hàm này thì tất cả các thread trong process sẽ bị hủy. Theo đó dẫn đến process bị hủy và chỉ định mã thoát của process là fuExitCode được truyền vào khi gọi hàm ExitProcess. **Lưu ý** : *tất cả những mã nguồn nằm sau hàm ExitProcess này sẽ không được thực thi.*

Không nên dùng hàm này vì rất dễ dẫn đến hệ thống không được dọn dẹp sạch sẽ

Ví dụ

#include <stdio.h>

#include <windows.h>

class CSomeObj {

public:

CSomeObj() {printf("Constructor\n");}

~CSomeObj() {printf("Destructor\n");}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

CSomeObj obj;

ExitProcess(0);

return 0;

}

Kết quả xuất ra mà hình là

Constructor

Ta thấy hàm ExitProcess làm process bị hủy ngay lập tức. Không kịp gọi các hàm destructor của các đối tượng

## Termianate Process

Hàm này cũng được dùng để hủy một proccess

BOOL TerminateProcess(

HANDLE hProcess,

UINT fuExitCode);

Hàm TerminateProcess khác ExitProcess ở một chỗ quan trọng

ExitProcess : thread gọi hàm này chỉ có thể hủy process chứa thread đó

TerminateProcess : thread gọi hàm này có thể hủy process chứa hoặc không chứa nó

Một process không thể ngăn chặn nó bị hủy

## When a Process Terminates

Tất cả các thread đang chạy của process cũng bị hủy theo

Tất cả các đối tượng người dùng và đối tượng GDI được cấp phát bởi process và tất cả các kernel objects đều bị hủy (Những kernel object này sẽ bị hủy nếu như không có một process nào mở hanle tới chúng. Tuy nhiên nếu vẫn còn process mở handle đến các ketnel objects này thì những object này sẽ không bị hủy.

Mã thoát của process chuyển từ STIL\_ACTIVE thành mã được truyền vào bởi hàm ExitProcess hoặc TerminateProcess.

Trạng thái của đối tượng process kernel process chuyển sang thành signaled. Những thread khác trong hệ thống của thể tự suppend cho đến khi process bị hủy

Biến đếm số lần sử dụng của đối tượng kernel object giảm xuống một

BOOL WINAPI GetExitCodeProcess(

\_\_in   HANDLE hProcess,

\_\_out  LPDWORD lpExitCode

);

Lấy về mà thoát của một ứng dụng

Nếu process còn sống và hàm được thực hiện thành công thì giá trị lpExitCode là STILL\_ACTIVE (259)

Nếu process đã chết và hàm được thì hiện thành công thì giá trị trà vể là một trong các giá trị sau

ExitCode trường truyền vào hàm ExitProcess hoặc TerminateProcess

Giá trị trả về từ hàm Main của process

Exception value của những exception không xử lý làm cho process bị hủy

Lưu ý : không sử dụng mã lỗi là 259 trùng với STILL\_ACTIVE

# Multi-Thread

WaitForSingleObject function

Hàm này return ( kết thúc không chờ nữa) khi một đối tượng được chờ chuyển sang trại thái signaled hay đã hết thời gian time-out. Một thread chuyển sang trạng thái singnaled khi thread đó kết thúc.

WaitForMultipleObjects

Cho phép chỉ định hàm return khi có 1 hoặc tất cả các đối tượng được signaled hay hết thời gian time-out

# Đồng bộ

CriticalSection

Dùng khi có một nhóm các câu lệnh mà có nhiều thread muốn thực thi mà ta không mún thread này chưa làm xong nhóm câu lệnh đó mà có thread khác nhảy vào.

InitializeCriticalSection : Chỉ gọi một lần để cấp phát tài nguyên cho đối tượng CriticalSection

EnterCriticalSection : Bất kì thread nào muốn thực thi nhóm câu lệnh cũng phải gọi lệnh này.

DeleteCriticalSection : Sau khi không dùng đối tượng CriticalSession thì giải phóng vùng nhớ

LeaveCriticalSection : Sau khi thực thi xong nhóm câu lệnh phải gọi hàm này để cho thread khác có thể nhảy vào

Event

Mỗi đối tượng event có số lần sử dụng, một giá trị boolean cho biết event là tự động reset hay phải reset thủ công và một giá trị boolean khác cho biết event là trạng thái signaled hay nonsignaled.

Event được sử dụng phổ biến trong trường hợp khi một thread đã thực hiện xong các công việc trong giai đoạn đầu sau đó phát tín hiệu cho một thread khác hoàn thành nốt các công việc còn lại. Sự kiện ban đầu được khởi tạo là nonsignaled, sau khi thread hoàn thành các công việc khởi tạo thì thread đó chuyển event sang trạng thái signaled. Tại thời điểm đó, một thread khác đang chờ trên event này, nhìn thấy event đã signaled thì thread đó sẽ được thực thi.

Manual-reset event

Khi event được signaled thì tất cả các thread đang chờ event này được thực thi.

Auto-rest event

Khi event được signaled thì chỉ một trong các thread đang chờ event này được thực thi.

Nếu một hàm chờ return nhờ vào một auto-rest event được signaled thì event đó sẽ tự động được chuyển sang nonsignaled.