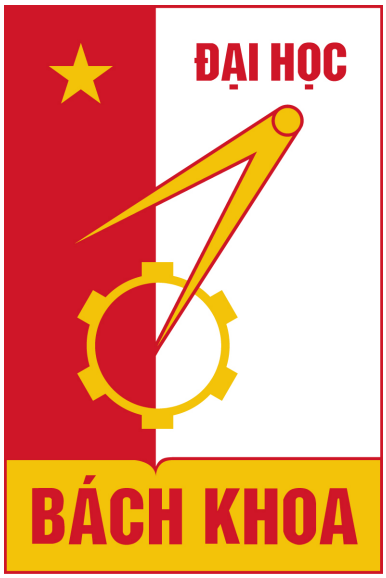
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

o0o



TinyShell sử dụng Win32 API

Báo cáo Project
Nguyên lý Hệ Điều Hành

Giáo viên hướng dẫn: TS. Phạm Đăng Hải

Sinh viên thực hiện : Nguyễn Tiến Long- 20180129
Phan Việt Hoàng- 20180086
Nguyễn Phi Phúc- 20180155
Bùi Đức Tuấn Dũng- 20180048
Phạm Trần Anh- 20180018

Lớp : CTTN CNTT K63

Hà Nội - 2020

Mục lục

Mở đầu

Công việc lập trình nhiều khi đòi hỏi các lập trình viên phải trực tiếp tham gia quản lý các file hệ thống, quản lý vào ra, quản lý tiến trình, hay nói cách khác là giao tiếp với hệ điều hành. Trên hệ điều hành Windows, Microsoft đã cung cấp các API để hỗ trợ người dùng thuận tiện hơn trong khi thực hiện các thao tác đó. Trong project này, nhóm chúng em sử dụng các giao thức được cung cấp trong Win32 API để xây dựng một TinyShell có thể thực hiện được một số lệnh cơ bản và một số thao tác với tiến trình và luồng

Chương 1

Tổng quan về Shell, TinyShell

1.1 Shell

Shell là một giao diện người dùng để truy nhập vào các dịch vụ của hệ điều hành. Một cách tổng quan, hệ điều hành sẽ cung cấp giao diện người dùng dòng lệnh(CLI) hoặc giao diện người dùng đồ hoạ(GUI), điều này tuy thuộc vào nhu cầu người dùng.

• Giao diện người dùng dòng lệnh yêu cầu người dùng phải quen thuộc với các lệnh và các cú pháp của lệnh, và cần hiểu được ngôn ngữ kịch bản của Shell, giao diện này thường chỉ phù hợp với các lập trình viện

• Giao diện ngườ dùng đồ hoạ là giao diện giúp người dùng dễ dàng giao tiếp, sử dụng hơn vì các yêu cầu được thực hiện trực quan, dễ hiểu. Tuy nhiên giao diện này khó đáp ứng được những thao tác đòi hỏi phải chọc sâu vào hệ thống.

1.2 TinyShell

TinyShell là một phiên bản "thu gọn" của Shell, cung cấp giao diện dòng lệnh. Khi được khởi chạy, TinyShell sẽ đọc yêu cầu(lệnh) trên màn hình console được người dùng nhập vào từ bàn phím. Sau đó, TinyShell sẽ phân tích lệnh và thực thi các thao tác đã được lập trình sẵn. Chức năng của TinyShell có thể chia làm 2 kiểu chính : đó là thực thị các tiến trính khác và thực thi các lệnh có sẵn trong shell(các lệnh này thường được gọi là built-in commands).

Trong quá trình xây dựng TinyShell, nhóm chúng em đã chia chương trình thành các module(header file) với các chức năng khác nhau để thuận tiện cho việc lập trình và quản lý chương trình .Chẳng hạn như processhandler.h, operate.h tương ứng với các chức năng về xử lý tiến trình, điều phối cách thực thi các lệnh đọc vào từ console. Chương trình của chúng em gồm các chức năng sau:

1. Các thao tác với tiến trình: liệt kê tiến trình, tìm kiếm pid của tiến trình, suspend, resume các tiến trình, mở các tiến trình dưới dạng foreground và background, tắt các tiến trình đang chạy,...

2. Các lệnh built-in: thay đổi thư mục, xem ngày giờ, liệt kê thư mục, ...
3. Các thao tác với các biến môi trường : Liệt kê các biến môi trường, đặt và làm mới các biến môi trường

Chương 2

Các thao tác với TinyShell

2.1 Các thao tác với tiến trình

2.1.1 Liệt kê với tất cả các tiến trình

- Cú pháp :

$$>\mathbf{ps}\ -\mathrm{all}$$

- Chức năng: Hiển thị ra tất cả các tiến trình đang chạy, với các thông tin bao gồm tên tiến trình , định danh tiến trình (PID), định danh tiến trình cha (PPID)

Trong đó, để thực hiện được lệnh, ta có sử dụng các WIN32API Function: CreateToolhelp32Snapshot(), Process32First() và Process32Next(). Cụ thể, ta thực hiện HANDLE CreateToolhelp32Snapshot() trả về 1 đối tượng HANDLE nhằm chụp lại tất cả các tiến trình đang chạy trong hệ thống, tiếp theo sử dụng BOOL Process32First() để lấy thông tin của tiến trình đầu tiên và lưu trữ trong 1 structure PROCESSENTRY32. Sau đó, lần lượt lấy thông tin của những tiến trình còn lại.

Sơ lược về các hàm:

1. HANDLE CreateToolhelp32Snapshot(DWORD dwFlags, DWORD th32ProcessID), với th32ProcessID = 0, khi dwFlags = TH32CS_SNAPPROCESS
2. BOOL Process32First(HANDLE hSnapshot, LPPROCESSENTRY32 lppe)
3. BOOL Process32Next(HANDLE hSnapshot, LPPROCESSENTRY32 lppe)

2.1.2 Liệt kê tất cả tiến trình con của một tiến trình

- Cú pháp:

>**ps** −child tham_so_1

VD: >ps -child 6583

- Chức năng: Chương trình liệt kê các tiến trình con của tiến trình có PID được truyền vào trong câu lệnh (tham_o_1 là PID của tiếnt trình)

Để thực hiện lệnh, ta sử dụng các function của WIN32 API : *CreateToolhelp32Snapshot()*, *Process32First()*, *Process32Next()*. Đầu tiên *CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)* sẽ trả về 1 thẻ (Handle) tham chiếu đến các tiến trình đang chạy trong hệ điều hành. Sau đó ta sử dụng *Process32First()*, *Process32Next()* để duyệt qua các tiến trình và kiểm tra PPID của các tiến trình này có bằng *tham_so_1* không, in ra thông tin tiến trình này nếu điều kiện trên thoả mãn

2.1.3 Tạm dừng một tiến trình

- Cú pháp:

>**ps** −**suspend** tham_so_1

VD: >ps -suspend 9100

- Chức năng: Lệnh cần tham số truyền vào là định danh của tiến trình cần tạm ngừng. Nếu tạm ngừng thành công, tiến trình sẽ không thể tiếp tục thực thi cho tới khi ta resume nó.

Trong đó, để thực hiện được lệnh, ta có sử dụng các WIN32API Function: CreateToolhelp32Snapshot(), Thread32First() và Thread32Next(), HANDLE OpenThread(), DWORD SuspendThread(). Cụ thể, ban đầu, ta thực hiện HANDLE CreateToolhelp32Snapshot() trả về 1 đối tượng HANDLE nhằm chụp lại tất cả các luồng đang chạy trong hệ thống, tiếp theo sử dụng BOOL Thread32First() để lấy thông tin của tiến trình đầu tiên và lưu trữ trong 1 structure THREADENTRY32. Sau đó, lần lượt với những luồng còn lại. Với từng luồng, ta thực hiện OpenThread() để truy cập tới luồng, sau đó suspend từng luồng bằng hàm SuspendThread(). Tóm lại, để tạm ngừng 1 tiến trình, ta sẽ tạm ngừng tất cả các luồng trong tiến trình đó.

Sơ lược về các hàm:

1. HANDLE CreateToolhelp32Snapshot(DWORD dwFlags, DWORD th32ProcessID), với th32ProcessID = 0, khi dwFlags = TH32CS_SNAPTHREAD
2. BOOL Thread32First(HANDLE hSnapshot, LPTHREADENTRY32 lppe)
3. BOOL Process32Next(HANDLE hSnapshot, LPTHREADENTRY32 lppe)
4. HANDLE OpenThread(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwThreadId)
5. DWORD SuspendThread(HANDLE hThread): trả về số lần luồng đã tạm ngừng nếu hàm thực hiện thành công, ngược lại trả về -1.)

2.1.4 Tiếp tục một tiến trình đang bị tạm dừng

- Cú pháp:

$$>\textbf{ps} -\text{resume tham\_so\_1}$$

VD: >ps -resume 9100

- Chức năng: Lệnh cần tham số truyền vào là định danh của tiến trình đang bị tạm ngừng cần tiếp tục. Tiến trình sẽ tiếp tục thực thi nếu lệnh thực hiện thành công.

Trong đó, để thực hiện được lệnh, ta có sử dụng các WIN32API Function: CreateToolhelp32Snapshot(), Thread32First() và Thread32Next(), HANDLE OpenThread(), DWORD ResumeThread(). Tương tự như với lệnh suspend, để tiếp tục 1 tiến trình, ta sẽ tiếp tục tất cả các luồng trong tiến trình đó.

Sơ lược về các hàm:

1. DWORD ResumeThread(HANDLE hThread): trả về số lần luồng đã tạm ngừng nếu hàm thực hiện thành công, ngược lại trả về -1.)

2.1.5 Liệt kê các luồng của một tiến trình

- Cú pháp:

$$>\textbf{ps} -\text{thread tham\_so\_1}$$

VD: >ps -thread 9100

- Chức năng: Cũng gân tương tự như lệnh liệt kê các tiến trình con, lệnh này liệt kê các luồng của 1 tiến trình với PID được truyền vào($tham\_so\_1$)

Không chỉ giống nhau về chức năng mà cách thực hiện của lệnh này với lệnh liệt kê các tiến trình cũng giống nhau, chỉ khác ở chỗ ta truyền tham số $TH32CS_SNAPTHREAD$ vào hàm *CreateToolhelp32Snapshot()* và sử dụng các hàm tương ứng với luồng *Thread32First()*, *Thread32Next()*

2.1.6 Tìm kiếm PID của một tiến trình

- Cú pháp:

$$>\textbf{ps} -\text{find tham\_so\_1}$$

VD: >ps -find chrome.exe

- Chức năng: Lệnh này cho phép ta tìm kiếm 1 tiến trình với tên của nó được truyền vào $tham\_so\_1$, TinyShell in ra console PID và PPID củ tiến trình tìm được (không in ra gì nếu không tìm ra)

Cũng giống như những lệnh tìm, liệt kê tiến trình luồng ở trên, ta cũng duyệt qua tất cả các tiến trình và kiểm tra theo tên các tiến trình tìm được.

2.1.7 Kết thúc một tiến trình

- Cú pháp:

```
>ps -kill tham_so_1
```

VD: >ps -kill 9100

- Chức năng: Kết thúc 1 tiến trình đang chạy dựa trên định danh của nó

Trong đó, để thực hiện được lệnh, ta có sử dụng các WIN32API Function: HANDLE OpenProcess(), BOOL TerminateProcess(). Cụ thể, ban đầu ta thực hiện truy cập tới tiến trình cần kết thúc thông qua hàm OpenProcess(), sau đó ta kết thúc tiến trình đó bằng việc sử dụng hàm TerminateProcess().

Sơ lược về các hàm:

1. HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)
2. BOOL TerminateProcess(HANDLE hProcess, UINT uExitCode): trả về 0 nếu thất bại, ngược lại trả về giá trị khác 0.)

2.1.8 Mở một tiến trình dưới dạng foreground

- Cú pháp:

```
>ps -fore tham_so_1
```

VD: >ps -fore PowerShell.exe

- Chức năng: Lệnh cho phép người dùng mở một tiến trình ở chế độ foreground. Tham số của tiến trình truyền vào có thể là tên ứng dụng cần chạy, hoặc đường dẫn tuyệt đối của tiến trình.

Lệnh được thực thi bàng cách sứ dụng các function cảu WIN32 API : *CreateProcess()* và *WaitForSingerObject()*.

2.1.9 Mở một tiến trình dưới dạng background

- Cú pháp:

```
>ps -back tham_so_1
```

VD: >ps -back PowerShell.exe

- Chức năng: Lệnh này tương tự lệnh ở chế độ foreground, khác ở tham số lệnh là $-back$

Lệnh được thực thi gần tương tự như lệnh trên, tuy nhiên tiến trình được gọi sẽ chạy trên 1 console khác và tham số thời gian trong *WaitForSingerObject()* là $INFINITE$.

2.2 Các thao tác với các lệnh built-in

2.2.1 Thay đổi đường dẫn

- Cú pháp:

```
>cd tham_so_1
```

VD: >cd D:\Program Files\Games\Pikachu

- Chức năng: Thay đổi đường dẫn thư mục hiện tại của tiến trình, nói cách khác, chuyển hướng sang một thư mục khác trong máy tính của bạn. Bạn có thể chuyển đến thư mục cha bằng cách nhập '..', thư mục hiện tại '.'

Windows API cho phép lấy và thay đổi thư mục hiện tại bằng các hàm *GetCurrentDirectory()* và *SetCurrentDirectory()*.

2.2.2 Liệt kê file và folder trong thư mục hiện tại

- Cú pháp:

```
>dir
```

- Chức năng: Liệt kê tất cả các files và các thư mục con ở trong thư mục hiện tại theo thứ tự bảng chữ cái.

2.2.3 In ra thời gian và ngày hiện tại

- Cú pháp:

```
>time
>date
```

- Chức năng: Hiển thị ngày và giờ hiện tại

2.3 Các thao tác với các biến môi trường

Biến môi trường là biến mô tả môi trường trong đó các chương trình chạy. Trong Windows, các biến môi trường có tên và giá trị . Ví dụ, biến windir (viết tắt của "Windows directory") có thể có giá trị "C: Windows" hoặc đường dẫn khác nơi Windows được cài đặt. Mỗi tiến trình đều có một khối các biến môi trường.

2.3.1 Xem tất cả các biến môi trường

- Cú pháp:

```
>enva -all
```

- Chức năng: Hiển thị tất cả các biến môi trường của tiến trình, cũng đồng thời là các biến môi trường được đặt trong máy tính.

Windows API cho phép lấy tất cả các biến môi trường của tiến trình gọi hàm *GetEnvironmentString()* và trả về một xâu chứa các biến môi trường có định dạng như sau:

Var1=Value1\0
Var2=Value2\0
Var3=Value3\0
...
VarN=ValueN\0\0

2.3.2 Lấy một biến môi trường cho trước

- Cú pháp:

```
>enva -get tham_so_1
```

VD: >enva -get Path

- Chức năng: Hiển thị các biến môi trường của tiến trình dựa trên tên biến môi trường, cũng đồng thời là các biến môi trường được trong máy tính

Windows API Function DWORD GetEnvironmenVariable(LPCTSTR name, LPTSTR buffer, DWORD size) lấy biến môi trường cho trước của tiến trình gọi hàm GetEnvironmenVariable() và trả về số lượng kí tự chứa trong buffer. Trong đó, hàm GetEnvironmenVariable(LPCTSTR name, LPTSTR buffer, DWORD size) có:

1. LPCTSTR name: con trỏ trỏ tới tên biến môi trường
2. LPTSTR buffer: con trỏ trỏ tới 1 buffer chứa giá trị biến môi trường
3. DWORD size: Kích thước của buffer.

2.3.3 Thiết lập hoặc thay đổi một biến môi trường

- Cú pháp:

```
>enva -set tham_so_1
```

VD: >enva -set x_var x_value

- Nếu biến môi trường chưa tồn tại trước đó, Windows API Function BOOL SetEnvironmentVariable(LPCTSTR name, LPCTSTR value) sẽ tạo ra 1 biến môi trường mới vào tập các biến môi trường. Còn nếu biến đã tồn tại, thì thay đổi lại giá trị của biến môi trường (x_value). Chú ý rằng, việc thay đổi biến môi trường chỉ tác động trên tập các biến môi trường của tiến trình gọi hàm, không làm thay đổi tập các biến môi trường của hệ thống.

Hàm SetEnvironmentVariable() trả về 0 nếu không thay đổi được, trả về giá trị khác 0 nếu ngược lại. Trong đó, hàm BOOL SetEnvironmentVariable(LPCTSTR name, LPCTSTR value):

1. LPCTSTR name: tên biến
2. LPCTSTR value: giá trị biến cần thay đổi.

Chương 3

Tổng kết

Qua bài báo cáo, thông qua việc xây dựng thiết kế một Shell đơn giản, ta có thể hiểu hơn về cách mà hệ điều hành quản lý một tiến trình, đồng thời thông qua việc sử dụng các hàm WIN32API, ta có cái nhìn tổng quan hơn về API trong Windows, và việc vận dụng nó trong việc thiết kế các ứng dụng sao cho hiệu quả và chính xác nhất.

Tài liệu tham khảo

[1] Programming reference for the Win32 API
https://docs.microsoft.com/en-us/windows/win32/api/

[2] CreateToolhelp32Snapshot function
https://docs.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot

[3] Tutorial - Write a Shell in C
https://brennan.io/2015/01/16/write-a-shell-in-c/?fbclid=IwAR3Pjy3D1t3ilX8h_UpvMbFxIe2MqxPpm-EtaaxzLcGn38NscbhnmK_O8Oo

[4] https://www.cems.uwe.ac.uk/~irjohnso/coursenotes/lrc/system/shell/cs3.htm?fbclid=IwAR2q85UYdD_qN3a7czdTeL_mGYFTYT5miPm1eZXYtBCrpKRAtxk5dxDzJyQ

[5] Tiny Shell (tsh) Implementation Overview
https://condor.depaul.edu/glancast/374class/docs/slides/Apr09/slide9.html?fbclid=IwAR3X3TtrAoxudir3cBUiPIsmsT3Pr2FLIW8xjk8b-QpSzuta7g2BhnzlTHk