Tiny Shell

Tạ Hữu Bình, Trần Trọng Hiệp, Lê Huy Hoàng, Võ Thục Khánh Huyền

Ngày 5 tháng 6 năm 2021

Tóm tắt nội dung

Bài báo cáo trình bày về các câu lệnh cơ bản trong Tiny Shell. Với mỗi câu lệnh cụ thể, ta sẽ tìm hiểu cách thức cài đặt (thư viện sử dụng, các hàm được cung cấp bởi Win32 API) và cách sử dụng câu lệnh đó. Cũng qua bài viết giúp chúng ta có thêm hiểu biết về các API quản lý tiến trình trong Windows và cách thức hoạt động của shell.

Mục lục

1 Giới thiệu chung về Tiny Shell

Cũng như Shell truyền thống, Tiny Shell cung cấp những chức năng cơ bản cho phép người dùng tạo, dừng, hủy tiến trình, thực thi file cũng như là xem địa chỉ của thư mục mình đang ở. Để giúp người dùng hiểu rõ về cơ chế hoạt động, Tiny shell đã cung cấp câu lệnh **help**, được mô tả theo cú pháp như sau:

help

```
help
```

Màn hình của người dùng sẽ hiện thị như sau:

1.	help	Provide Help information for Windows commands
2.	cd ..	Change to the parrent directory of the current directory
3.	cd 'path'	Change current directory to this path
4.	dir	Display list of files in parent directory
5.	date	Display date
6.	time	Display time
7.	clock	Clear screen and display clock
8.	countdown	Display countdown clock, add fore or back mode, example: countdown fore, countdown back
9.	notepad	Open system notepad, add fore or back mode, example: notepad fore, notepad back
10.	stop 'ID'	Stop a running process
11.	resume 'ID'	Resume a stopping process
12.	kill 'ID'	Kill a running process
13.	kill -1	Kill all running processes
14	list	Display list of processes
15	[*.bat]	Read *.bat file and run list of command lines
16	path of *.exe	Run *.exe file, add fore or back mode
17	env a	Display the value of the environment variable a, example: env path If a = null, display all the environment variables and their values
18	addenv (a) b	Add the environment variable a with its value b
19	delenv a	Delete the environment variable a
20	history	Print all used commands
21	clean	Clean the history
22	clear	Clear tiny shell
23	exit	Exit process

2 Các lệnh quản lý tiến trình

2.1 background và foreground

- background: 'lệnh' back.
  Ví dụ: countdown back
- foreground: 'lệnh' fore.
  Ví dụ: countdown fore

Để tạo một tiến trình ta dùng hàm **CreatePocess(...)**. Khi một tiến trình được khởi tạo, nếu ta muốn tiến trình đó chạy ở chế độ foreground, ta sẽ dùng lệnh **WaiForSingleObject(...)** sau khi khởi tạo tiến trình. Khi chạy foreground ta sẽ phải kết thúc tiến trình rồi mới có thể tiếp tục nhập lệnh tiếp theo. Ngược lại, nếu ta muốn chương trình đó chạy ở chế độ background, ta sẽ không dùng lệnh đó nữa và chúng ta có thể tiếp tục nhập lệnh kế tiếp.

```
1    WaitForSingleObject(hProcess, INFINITE);
```

2.2 list

- **list** : hiển thị danh sách các trình con đang chạy

Lệnh "list" được gọi khi chúng ta muốn nhận biết được danh sách các tiến trình đang chạy. Ban đầu ta có một biến đếm n, khởi tạo $n = 0$, khi ta khởi tạo 1 tiến trình, nếu tiến trình đó được chạy dưới dạng foreground, khi tiến trình đó kết thúc, ta mới quay trở lại để có thể tiếp tục làm việc được với Tiny Shell, nên khi ta dùng lệnh "list" thì tiến trình đó đã bị đóng và không có trong chương trình đang chạy, ta giữu nguyên biến n. Ngược lại khi ta chạy tiến trình đó chạy dưới dạng background, ta sẽ tăng biến đếm của n lên 1 để kiểm soát được số lượng các tiến trình.

Mảng các **PROCESS_INFORMATION** sẽ cho chúng ta biết thông tin về id, thẻ tiến trình bị kết thúc đóng và thẻ đối tượng của tiến trình. Mảng **cString** sẽ cho ta biết tên của các tiến trình và mảng **status** sẽ cho ta biết trạng thái của tiến trình.

Khi in ra thông tin các tiến trình đang chạy, sẽ dùng vòng for từ 1 đến n. Trong mỗi lần lặp ta sẽ sử dụng biến**DWORD**và hàm **BOOL GetExitCodeProcess(HANDLE hProcess, LPDWORD lpExitCode)** để kiểm tra xem liệu tiến trình đó đã đóng chưa, nếui giá trị của **DWORD** trả về khác với 259 tức là ứng dụng đã thoát và chúng ta tiến hành loại bổ nó ra khỏi danh sách list, nếu không ta sẽ in ra danh sách tiến trình đang chạy.

```
1    DWORD dwExitCode;
2    GetExitCodeProcess(pi[i].hProcess, &dwExitCode);
3    if (dwExitCode != 259) {
4        TerminateProcess(pi[i].hProcess, 0);
5        CloseHandle(pi[i].hThread);
6        CloseHandle(pi[i].hProcess); }
```

2.3 kill

- **kill 'id'** : kill 1 tiến trình với id tương ứng
- **kill -1** : kill tất cả các tiến tình

Khi ta muốn dừng một tiến trình bằng id, ta sẽ sử dụng lệnh *kill*. Bằng cách dựa vào thông tin về id của các tiến tình trên bảng list, ta sẽ có cú pháp **kill 'id'** để *kill* tiến trình có id tương ứng. Nếu chúng ta muốn *kill* tất cả các tiến trình, chúng ta sẽ sử dụng lệnh **kill -1**.

Khi hàm *kill* 'id' được gọi, ta sẽ dùng vòng for từ 1 đến n để tìm tiến trình với id tương ứng, sau đó ta sẽ tiến hành thủ tục *kill* tiến trình. Nếu chúng ta không tìm được tiến trình với id tương ứng, ta sẽ thông báo ra lỗi.

Sau khi *kill* 1 tiến trình xong, ta sẽ giảm biến n đi 1, sau đó ta dùng 1 vòng for từ vị trí tiến trình bị *kill* đến $n-1$ để gán thẻ thông tin của tiến trình đằng trước bằng thẻ thông tin của tiến trình ngay sau nó, thiết lặp lại giá trị của flag để biết được rằng đã tìm được id tương ứng, rồi ta break khỏi vòng for và chờ đợi lệnh tiếp theo.

Nếu ta dùng lệnh *kill* tất cả các tiến trình, ta vẫn sẽ dùng vòng for từ 1 đến n và trong mỗi vòng for ta *kill* cá tiến trình tương ứng, sau khi thoát khỏi vòng for ta set $n = 0$ rồi chờ đợi lệnh tiếp theo.

kill

```
1  TerminateProcess(pi[i].hProcess, 0);
2  CloseHandle(pi[i].hThread);
3  CloseHandle(pi[i].hProcess);
```

2.4 stop và resume

- **stop** : tạm dừng một tiến trình đang chạy
- **resume** : tiếp tục một tiến trình đã tạm dừng

Về phương pháp triển khai, 2 lệnh trên có phương pháp tương tự giống như lệnh **kill 'id'**. Tuy nhiên trong mỗi vòng for, khi tìm được tiến trình tương ứng, ta sẽ không giảm n đi 1 mà thay vào đó là đánh dấu lại trạng thái của tiến trình. Khi 1 triến trình bị stop, status của nó bằng 0, và khi tiến trình đó được *resume* trở lại, *status* của nó bằng 1.

stop

```
1  SuspendThread(pi[i].hThread);
```

resume

```
1  ResumeThread(pi[i].hThread);
```

3 Các lệnh đặc biệt

3.1 time, clock và date

Đây đều là các lệnh in thời gian, sử dụng thư viện "ctime.h". Giữa hai lệnh này có các sự khác biệt căn bản sau:

- *time*: Chỉ in thời gian theo giờ, phút, giây.
- *clock*: Xóa màn hình, đồng thời in thời gian theo giờ, phút, giây nhưng kèm với hình ảnh biểu tượng của một chiếc đồng hồ.
- *date*: In thời gian kèm với ngày hiện tại.

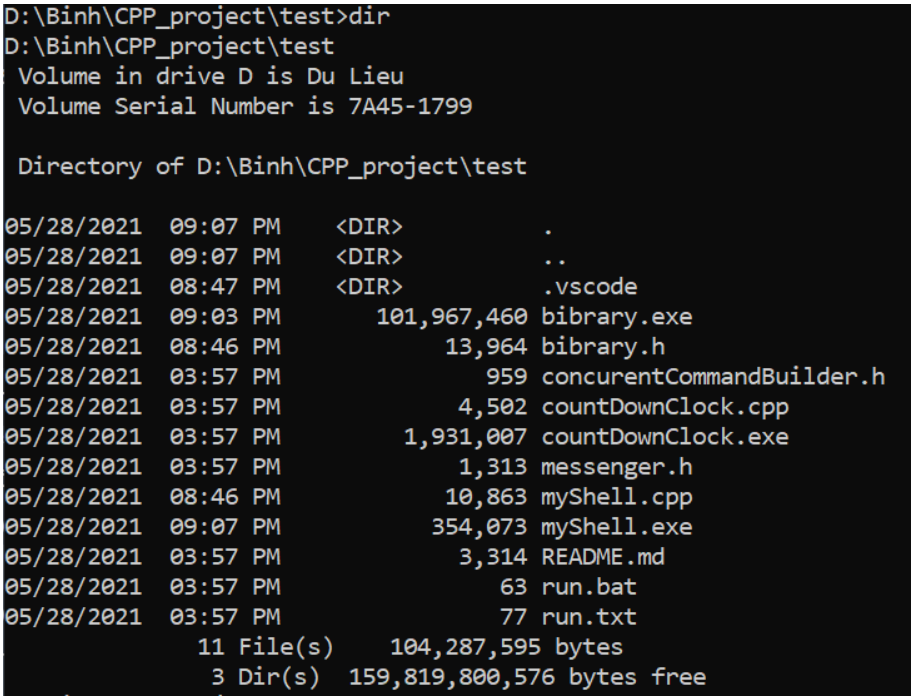
Cả ba lệnh đều được thực hiện thông qua các luồng, do đó *cin.get()* được sử dụng để có thể kết thúc việc in ra thời gian. Cụ thể, người dùng sẽ nhấn "enter" để có thể thực hiện các câu lệnh tiếp theo trong shell.

3.2 dir và cd

Đây là hai câu lệnh liên quan đến địa chỉ, đều tận dụng các hàm cung cấp bởi *win32 api*, riêng lệnh cd còn hỗ trợ bởi thư viện "unistd.h".

- Với lệnh *dir*, chương trình sẽ in ra địa chỉ làm việc hiện thời. Hàm *_getcwd* trả về giá trị con trỏ của một buffer (chứa địa chỉ làm việc). Giá trị trả về NULL ám chỉ lỗi đã xảy ra.Trong trường hợp lấy được buffer mà không xảy ra lỗi, hàm *_chdir* sẽ được sử dụng để kiểm tra lại xem địa chỉ có hợp lệ hay không. Nếu hợp lệ sẽ kết hợp câu lệnh của hệ thống *system("dir")* để in ra địa chỉ hiện thời cùng các folder, file bên trong.
- Với lệnh *cd*:
  - Khi thực hiện "cd ..", địa chỉ sẽ được đưa lên một cấp.
  - Khi thực hiện "cd <*folder name*>", nếu tên folder mà hợp lệ thì địa chỉ sẽ được chuyển đến folder đó thông qua hàm *chdir* của thư viện "unistd.h", ngược lại sẽ báo cho dùng biết nhập tên folder bị lỗi.
  - Khi thực hiện "cd <*file name*>" (*file name* phải có đủ cả đuôi file), nếu tên file mà hợp lệ và có định dạng mà hệ điều hành có thể đọc được trực tiếp thì file sẽ được mở mặc định trong background mode , ngược lại sẽ báo lỗi.

Ví dụ, khi yêu cầu Tiny Shell thực hiện lệnh dir, ta thu được kết quả như sau:



Bên cạnh địa chỉ làm việc hiện tại và các folder, file bên trong, *system("dir")* còn cung cấp cho người dùng các thông tin như tên và số hiệu của ổ đĩa chứa địa chỉ làm việc, số lượng file,... Ở đây, giả sử muốn mở file myShell.exe, người dùng sẽ phải thực hiện lệnh:

cd

```
cd myShell.exe
```

Nếu không có đuôi file, hoặc đuôi là .cpp thì Tiny Shell đều không thể mở được.

3.3 clear và exit

clear đơn giản dùng câu lệnh *system("cls")* để xóa màn hình trong shell. *exit* dùng câu lệnh *exit(0)* để tắt chương trình. Tuy nhiên trước đó, hàm *sleep_for* của namespace "this_thread" trong C++ sẽ làm chương trình bị trễ 1.5 giây để kịp hiện lời chào cũng như thông báo xóa các tiến trình con đang làm việc.

3.4 history và clean

Lệnh *history* được dùng để in ra các câu lệnh đã thực hiện. Để làm điều này, class *vector* sẽ được tận dụng vì đặc tính như một mảng không bị giới hạn bởi kích thước khai báo trước. Phương thức *push_back* có tác dụng như "push" trong cấu trúc dữ liệu "stack", tiện lợi cho việc lưu trữ các lệnh mới. Phương thức *clear* cũng được sử dụng để "dọn sạch" lịch sử, thể hiện qua lệnh *clean*.

4 Biến môi trường

4.1 env

Có 2 loại biến môi trường là biến môi trường người dùng và biến môi trường hệ thống. Trong Tiny Shell, biến môi trường được gọi ra là biến môi trường người dùng. Có hai phương thức xem giá trị của biến môi trường:

- Xem giá trị của một biến môi trường cụ thể:

```
env name
```

  Trong đó: "name" là tên biến mà người dùng cần xem giá trị.

- Liệt kê tất cả các biến môi trường và giá trị tương ứng của nó:

```
env
```

4.2 addenv

Để thêm giá trị cho một biến môi trường, người dùng thực hiện câu lệnh sau:

addenv

```
addenv (name) value
```

Trong đó:

- "name" là tên biến môi trường
- "value" là giá trị cần thêm
- Dấu ngoặc đơn là cần thiết để ngăn cách giữa tên biến và giá trị

Giá trị mới của biến sẽ được thêm vào đuôi của giá trị cũ, ngăn cách bởi dấu ";".
Nếu biến đó chưa tồn tại, Tiny Shell sẽ tạo một biến mới có giá trị tương ứng.

4.3 delenv

Để xóa một biến môi trường, người dùng thực hiện câu lệnh sau:

delenv

```
delenv name
```

Trong đó: name là tên biến môi trường cần xóa.

5 Các lệnh liên quan đến file

5.1 Đọc file *.bat

Tiny Shell có chức năng đọc file *.bat và thực thi lần lượt từng câu lệnh trong file đó (mỗi lệnh nằm trên một dòng), file *.bat này nằm trong cùng thư mục với chương trình Tiny Shell.

Cú pháp và ví dụ như sau:

file *.bat

```
file_bath_name

run.bat
```

Một file *.bat có cấu trúc tương tự như ví dụ sau:

file *.bat

```
c:/windows/system32/notepad.exe fore
countdown back
dir
help
```

5.2 Thực thi file *.exe

Tiny Shell có thể thực thi các file *.exe thông qua đường dẫn đến file đó, có thể chọn mode khi file thực thi, backgound hoặc foreground.

Cú pháp và ví dụ như sau:

file *.exe

```
file_exe_path mode

c:/windows/system32/notepad.exe fore
c:/windows/system32/notepad.exe back
```

6 Tổng kết

Quá trình làm Tiny Shell đã giúp nhóm chúng em hiểu hơn về cách hoạt động của tiến trình trong hệ thống cũng như là cách sử dụng WIN32 API [1]. Trong tương lai, nhóm sẽ bổ sung thêm nhiều chức năng của Shell truyền thống vào Tiny Shell của nhóm.

Để hoàn thành được báo cáo này, nhóm chúng em xin tỏ lòng biết ơn sâu sắc đến thầy Phạm Đăng Hải, cảm ơn thầy vì những tiết dạy tuyệt vời và đáng quý của thầy ở môn Nguyên lý hệ điều hành.

Toàn bộ source code cho Tiny Shell của nhóm được đặt ở đây: link.

Tài liệu

[1] Microsoft. *Win32 api*. https://docs.microsoft.com/en-us/windows/win32/.