ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

BÁO CÁO

NGUYÊN LÝ HỆ ĐIỀU HÀNH

CHỦ ĐỀ: Tiny Shell

Nhóm sinh viên thực hiện: Trần Tuấn Minh 20215618

Phan Trọng Cường 20215538

Lớp: CTTN Khoa học Máy tính – K66

Giảng viên hướng dẫn: TS. Phạm Đăng Hải

HÀ NỘI - 2023

Lời mở đầu

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

Mục Lục

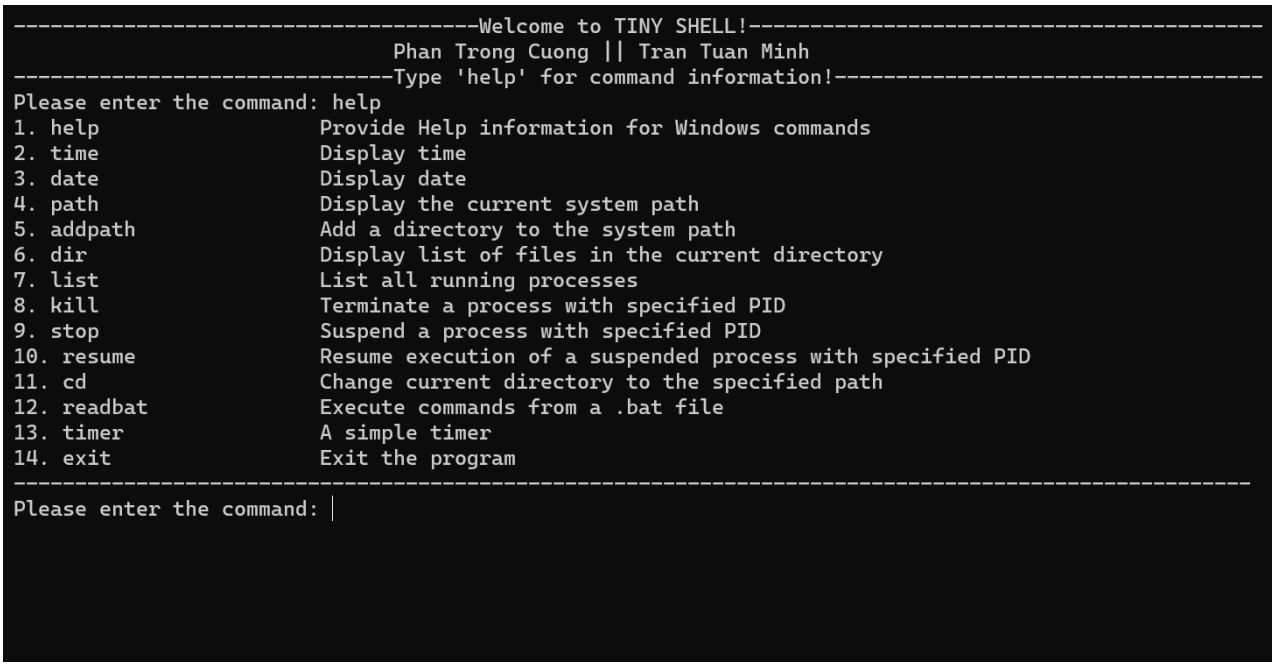
1 Giới thiệu chung

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.

Nhằm giúp người dũng hiểu rõ về cơ chế hoạt động, Tiny shell cung cấp câu lệnh help giúp người dùng có thể dễ dàng sử dụng với các câu lệnh như sau:

help

Màn hình người dùng sẽ hiển thị:



2 Các lệnh quản lý tiến trình

2.1 BackgroundProcess và **ForegroundProcess**

Hai hàm **ForegroundProcess** và **BackgroundProcess** được sử dụng để chạy một tiến trình trong chế độ tiến trình nền hoặc tiến trình tiền mặt.

Hàm ForegroundProcess được sử dụng để chạy một tiến trình trong chế độ tiến trình tiền mặt. Nó tạo một tiến trình mới bằng cách sử dụng hàm **CreateProcess** và chờ đến khi tiến trình kết thúc bằng cách sử dụng hàm **WaitForSingleObject**. Trong quá trình chờ, tiến trình mới sẽ chạy trong cửa sổ dòng lệnh hiện tại.

Hàm **BackgroundProcess** cũng tạo ra một tiến trình mới bằng cách sử dụng hàm **CreateProcess**, nhưng nó cho phép tiến trình chạy trong chế độ tiến trình nền. Sau khi tiến trình được tạo, nó được lưu trữ trong một mảng các tiến trình process để theo dõi và quản lý.

2.2 list

Hàm **ListProcesses** được sử dụng để liệt kê thông tin về tất cả các tiến trình đang chạy trong hệ thống.

Đầu tiên, hàm tạo một "**snapshot**" (bản chụp) của tất cả các tiến trình trong hệ thống bằng cách sử dụng hàm **CreateToolhelp32Snapshot**. Nếu việc tạo snapshot không thành công, hàm sẽ thông báo lỗi và kết thúc.

Sau đó, hàm lấy thông tin về tiến trình đầu tiên trong snapshot bằng cách sử dụng hàm Process32First. Nếu việc lấy thông tin không thành công, hàm sẽ thông báo lỗi và kết thúc.

Tiếp theo, hàm in ra thông tin về tiến trình, bao gồm tên tiến trình, ID tiến trình và ID tiến trình cha. Mỗi thông tin được căn lề và in ra theo định dạng.

Sau đó, hàm sử dụng vòng lặp **do-while** và hàm **Process32Next** để lặp qua tất cả các tiến trình trong snapshot và in ra thông tin của chúng.

Cuối cùng, hàm đóng handle của snapshot bằng cách sử dụng hàm **CloseHandle**, kết thúc quá trình liệt kê tiến trình.

2.3 kill

Hàm **KillProcess** được sử dụng để kết thúc một tiến trình (process) đang chạy bằng cách sử dụng hàm TerminateProcess. Tóm tắt chức năng của hàm:

- Nhận đối số pid (Process **ID**) của tiến trình cần kết thúc.
- Mở tiến trình thông qua hàm **OpenProcess** với quyền PROCESS_**TERMINATE** để có thể kết thúc tiến trình.
- Kiểm tra xem việc mở tiến trình có thành công hay không. Nếu không, hiển thị thông báo lỗi và kết thúc hàm.

- Sử dụng hàm **TerminateProcess** để kết thúc tiến trình. Nếu thành công, hiển thị thông báo thành công. Ngược lại, hiển thị thông báo lỗi.
- Đóng handle của tiến trình thông qua hàm **CloseHandle**.

Tóm lại, hàm này cho phép kết thúc một tiến trình bằng cách sử dụng Process ID của tiến trình đó.

```
HANDLE hProcess = OpenProcess(PROCESS_TERMINATE, FALSE, pid);

CloseHandle(hProcess);
```

2.4 stop và resume

'stop' gọi ra hàm SuspendProcess được sử dụng để đình chỉ (suspend) các luồng (threads) trong một tiến trình cụ thể. Dưới đây là mô tả chi tiết của từng bước trong hàm:

- Mở tiến trình: Hàm sử dụng OpenProcess để mở tiến trình với quyền truy cập PROCESS_SUSPEND_RESUME. Nếu việc mở tiến trình không thành công, thông báo lỗi sẽ được hiển thị và hàm sẽ kết thúc.
- Lấy snapshot của các luồng: Sử dụng CreateToolhelp32Snapshot để lấy snapshot của tất cả các luồng trong hệ thống. Nếu việc lấy snapshot không thành công, thông báo lỗi sẽ được hiển thị, và các handle đã mở sẽ được đóng trước khi hàm kết thúc.
- Lặp qua từng luồng: Hàm sử dụng Thread32First và Thread32Next để lặp qua từng luồng trong snapshot. Nếu luồng có th32OwnerProcessID trùng khớp với PID của tiến trình, hàm sẽ mở luồng đó bằng OpenThread với quyền truy cập THREAD_SUSPEND_RESUME.
- Đình chỉ luồng: Nếu mở luồng thành công, hàm sử dụng SuspendThread để đình chỉ luồng đó.
- Đóng handle và thông báo thành công: Sau khi lặp qua tất cả các luồng, các handle và tiến trình sẽ được đóng lại. Thông báo "Process suspended successfully" sẽ được hiển thị.

Hàm SuspendProcess cho phép đình chỉ tạm thời các luồng trong một tiến trình, gián đoạn việc thực thi của các luồng này.

'resume' gọi hàm ResumeProcess được sử dụng để tiếp tục (resume) các luồng (threads) trong một tiến trình cụ thể. Dưới đây là mô tả chi tiết của từng bước trong hàm:

- Mở tiến trình: Hàm sử dụng OpenProcess để mở tiến trình với quyền truy cập PROCESS_SUSPEND_RESUME. Nếu việc mở tiến trình không thành công, thông báo lỗi sẽ được hiển thị và hàm sẽ kết thúc.
- Lấy snapshot của các luồng: Sử dụng CreateToolhelp32Snapshot để lấy snapshot của tất cả các luồng trong hệ thống. Nếu việc lấy snapshot không

thành công, thông báo lỗi sẽ được hiển thị, và các handle đã mở sẽ được đóng trước khi hàm kết thúc.

- Lặp qua từng luồng: Hàm sử dụng Thread32First và Thread32Next để lặp qua từng luồng trong snapshot. Nếu luồng có th32OwnerProcessID trùng khớp với PID của tiến trình, hàm sẽ mở luồng đó bằng OpenThread với quyền truy cập THREAD_SUSPEND_RESUME.
- Tiếp tục luồng: Nếu mở luồng thành công, hàm sử dụng ResumeThread để tiếp tục thực thi luồng đó.
- Đóng handle và thông báo thành công: Sau khi lặp qua tất cả các luồng, các handle và tiến trình sẽ được đóng lại. Thông báo "Process resumed successfully" sẽ được hiển thị.

Hàm ResumeProcess cho phép tiếp tục thực thi các luồng đã được đình chỉ trước đó trong một tiến trình, cho phép các luồng này tiếp tục thực hiện công việc của mình.

3 Các lệnh

3.1 time và date

Hai lệnh sử dụng thư viện “ctime” để in ra thời gian của hệ thống:

- Lệnh ‘time’ in ra thời gian theo giờ, phút, giây. Người dùng sử dụng phím ‘Enter’ để kết thúc đồng hồ và tiếp tục chương trình.
- Lệnh ‘date’ in ra ngày hiện tại.

3.2 dir

- Lệnh ‘dir’ in ra địa chỉ làm việc hiện tại.

Ta sử dụng hàm “_getcwd” để lấy đường dẫn thư mục hiện tại và lưu trữ vào buffer. Kiểm tra xem hàm “_getcwd” có thành công không. Nếu trả về giá trị NULL thì báo lỗi và kết thúc hàm, còn nếu không thì ta sử dụng hàm “system” để thực thi lệnh ‘dir’ hiển thị danh sách tập tin và thư mục trong thư mục làm việc hiện tại.

3.3 cd

- Lệnh ‘cd’ được sử dụng để thay đổi thư mục làm việc hiện tại trong hệ điều hành.

Lệnh nhận vào một ‘path’, đại diện cho đường dẫn tới thư mục mà chúng ta muốn chuyển tới. Sử dụng hàm “SetCurrrentDirectory” và truyền đường dẫn ‘path’ dưới dạng con trỏ chuỗi kí tự c_str(). Nếu hàm trả về 0 thì báo lỗi thay đổi thư mục không thành công, nếu khác 0 thì in ra thông báo cho biết thư mục làm việc hiện tại đã được thay đổi thành công.

3.4 path và addpath

- Lệnh ‘path’ được sử dụng để in ra các giá trị của biến môi trường trong hệ điều hành.

Sử dụng hàm “getenv” để lấy giá trị của biến môi trường được chỉ định và trả về một con trỏ đến chuỗi ký tự đại diện cho biến môi trường. Nếu trả về giá trị NULL thì in ra một chuỗi rỗng, nếu không, tức biến môi trường tồn tại, chúng ta lấy giá trị từ con trỏ và trả về dưới dạng một chuỗi.

- Lệnh ‘addpath’ được sử dụng để thêm một đường dẫn mới vào biến môi trường “PATH” trong hệ điều hành.

Sử dụng hàm “gentev” để lấy giá trị hiện tại của biến môi trường “PATH” và lưu vào biến ‘path’. Thêm đường dẫn mới ‘newPath’ vào biến ‘path’ bằng cách nối chuỗi. Sau đó, sử dụng hàm ‘_putenv’ để cập nhật biến môi trường “PATH” với giá trị mới.

3.5 readbat

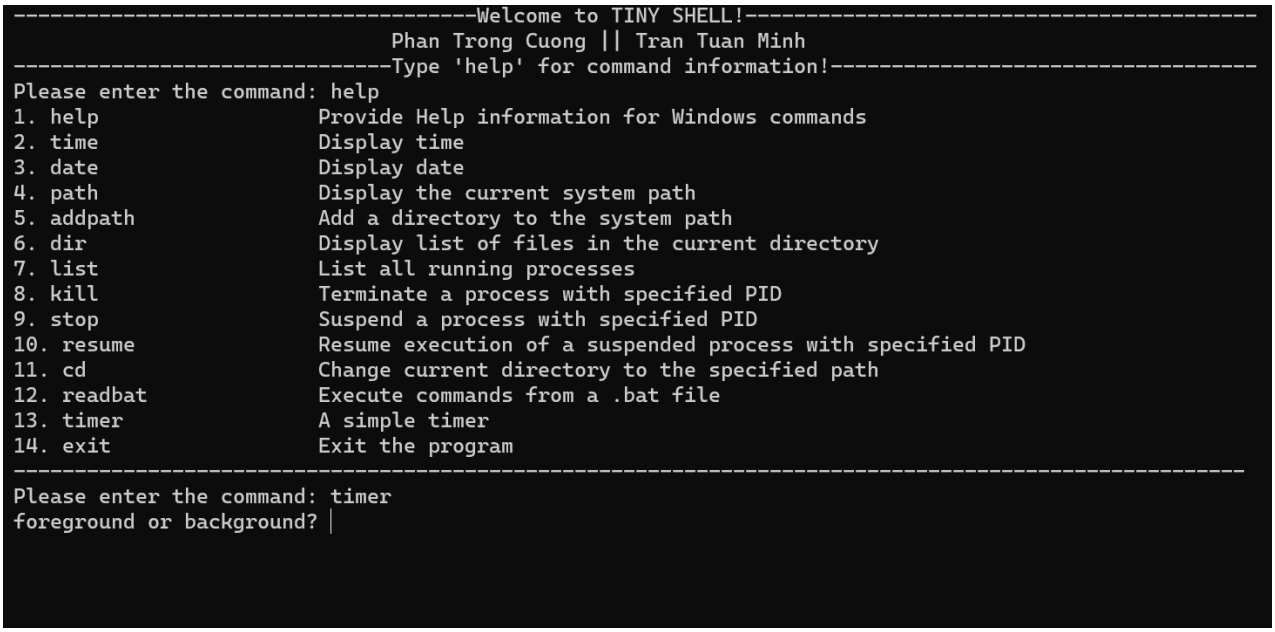
- Lệnh ‘readbat’ được sử dụng để đọc các file *.bat

File *.bat này phải nằm cùng thư mục với chương trình TinyShell. Khi nhập lệnh ‘readbat’, chương trình sẽ yêu cầu nhập một <filename> nếu filename có đuôi .bat và nằm cùng thư mục sẽ tiến hành đọc file, còn nếu không thì báo lỗi.

3.6 timer

- Lệnh ‘timer’ tạo ra một tiến trình đếm thời gian.

Chương trình sẽ yêu cầu nhập chế độ foreground hay background. Khi nhập ‘foreground’ bộ đếm thời gian sẽ được chạy ở chế độ tiến trình tiền mặt, còn nếu nhập ‘background’ bộ đếm thời gian sẽ được chạy ở trong nền. Ngoài ra ta có thể sử dụng lệnh Ctrl + C từ bàn phím để kết thúc tiến trình ‘timer’.



3.6 exit

- Lệnh ‘exit’ được sử dụng để thoát chương trình. Ngoài ra chúng ta cũng có thể sử dụng tổ hợp phím Ctrl + C để thoát chương trình.

4 Tổng kết

Qua quá trình thiết kế và cài đặt một *shell* đơn giản. Chúng em đã hiểu hơn về các API quản lý tiến trình trong Windows, hiểu được cách cài đặt và cách thức shell làm việc.

Chúng em xin gửi lời cảm ơn đến thầy Phạm Đăng Hải vì những bài giảng, tài liệu và các bài tập thực tế đã giúp chúng em hiểu hơn về bộ môn Nguyên lý Hệ điều hành.

5 Tài liệu tham khảo

- Bài giảng Nguyên lý Hệ điều hành, TS. Phạm Đăng Hải.
- learn.microsoft.com/en-us/windows/win32/shell/shell-entry
- learn.microsoft.com/vi-vn/visualstudio/ide/reference/shell-command?view=vs-2019