Trường đại học Khoa học Tự nhiên - Đại học Quốc gia thành phố Hồ Chí Minh

Khoa Công Nghệ Thông Tin

------oOo------

Báo cáo thực hành

Simple

Linux Shell



Giáo viên hướng dẫn: **Lê Giang Thanh – Nguyễn Thị Thanh Huyền**

Lớp: **Hệ điều hành - CQ2017/21**

Sinh viên thực hiện: **Âu Dương Tấn Sang – 1712145**

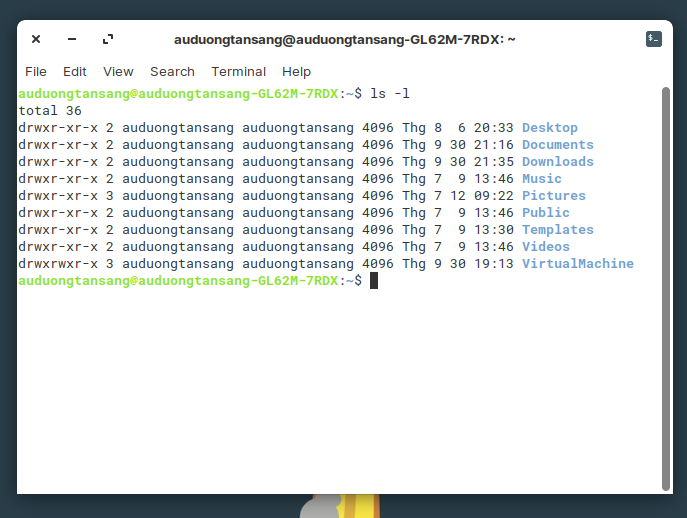
Github repository của bài thực hành này: <https://github.com/auduongtansang/SimpleShell>

HCM – 09/2019

**I. Giới thiệu:**

Shell là một chương trình dòng lệnh, cho phép người dùng nhập vào các lệnh mà hệ điều hành hỗ trợ và thực thi chúng. Mỗi hệ điều hành có rất nhiều chương trình shell của bản thân nhà sản xuất hệ điều hành và của cả bên thứ ba. Một số ví dụ điển hình: Command Prompt hay Windows PowerShell của hệ điều hành Windows, hay Terminal của các hệ điều hành nhân Linux,…

Mục tiêu của bài thực hành này là giúp sinh viên hiểu được các khải niệm cơ bản về cách mà hệ điều hành hoạt động, thông qua việc lập trình giao tiếp với hệ điều hành bằng các hàm hệ thống của ngôn ngữ C.

  
Terminal trên hệ điều hành Linux

**II. Tự đánh giá mức độ hoàn thành:**

Tạo process con, thực thi lệnh, concurrent 100%

Lưu và tải lại lịch sử 100%

IO redirect 100%

Communicate via pipe 100%

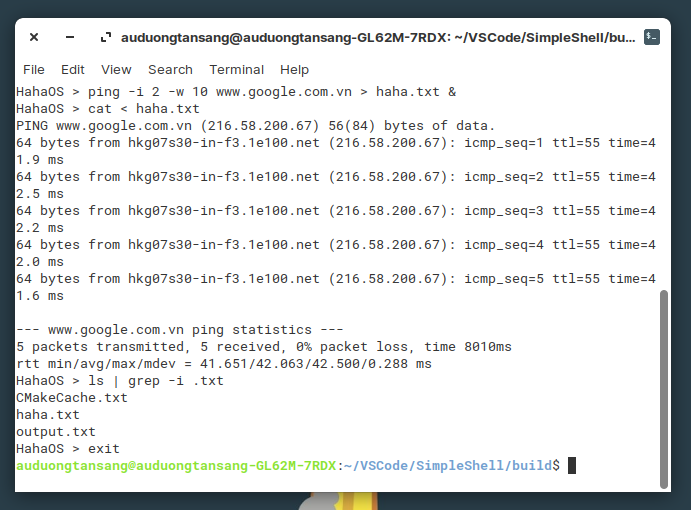
Hạn chế: chưa thực hiện được những câu lệnh có cấu trúc phức tạp như a | b | c | d hoặc

a & b hay a && b,... Tuy nhiên có thể chạy được dạng redirect + concurrently runnint như: command > file &.

**III. Tổng quan:**

Chương trình dừng lại ở dấu nhắc lệnh, đợi người dùng nhập vào một lệnh và nhấn phím enter, sau đó sẽ phân giải lệnh, tạo ra tiến trình con để thực thi lệnh đó. Người dùng có thể tùy chọn nhập lệnh tiếp theo ngay sau đó mà không phải chờ đợi, hoặc chờ đến khi tiến trình con thực thi xong lệnh rồi mới nhập lệnh tiếp theo.

Ngoài ra, chương trình còn hỗ trợ chạy lại câu lệnh vừa nhập ngay trước đó, hỗ trợ IO redirect và pipe. Hình dưới đây là một số ví dụ minh họa từ chương trình đã được viết xong.

  
Các chức năng cơ bản

**IV. Báo cáo cài đặt chương trình:**

Chương trình được viết theo cấu trúc hướng thủ tục, bằng các lệnh hết sức cơ bản của ngôn ngữ C, không dùng C++. Tổng thể, chương trình chạy như sau:

while (1) //Lặp đến khi break xảy ra

{

printf(“HahaOS> “); //Dấu nhắc lệnh

command = readCommand(); //Đọc lệnh từ bàn phím

/// Vùng code này dùng để xử những lệnh build-in

/// Ví dụ như exit, !!, ...

if (strcmp(command, “exit”) == 0) //Nhập exit để thoát

break;

///

///

args = parseCommand(command); //Phân giải lệnh

execCommand(args); //Thực thi lệnh

}

**a. Đọc lệnh:**

Thay vì dùng gets() sẽ có thông báo warning unsafe, chương trình dùng fgets() và bỏ đi kí tự ‘\n’ do bản chất của hàm này.

fgets(command, MAX\_LEN, stdin);

command[strlen(command) - 1] = ‘\0’;

**b. Lịch sử lệnh:**

Khi vừa khởi động chương trình, biến lưu lịch sử sẽ được gán bằng NULL. Mỗi khi người dùng nhập lệnh và bấm enter, command sẽ được sao chép vào biến lịch sử.

command = readCommand();

strncpy(history, command, strlen(command) + 1);

Và nếu nhập !!, lịch sử sẽ được xử lý và sao chép vào command thay thế cho chuỗi !!

/// Vùng code này dùng để xử những lệnh build-in

/// Ví dụ như exit, !!, ...

if (strcmp(command, “exit”) == 0) //Nhập exit để thoát

break;

else if (strcmp(command, “!!”) == 0) //Nhập !!

{

if (history == NULL)

{

printf(“No history\n”);

continue;

}

strncpy(command, history, strlen(history) + 1);

}

///

///

**c. Phân giải lệnh:**

Hàm parseCommand(command) sẽ tách char \*command thành những từ đơn (cách nhau bởi các khoảng trắng). Hàm trả về mảng kết quả char \*\*args, và thêm vào cuối mảng này một args NULL. Chương trình đã cài đặt dùng hàm strtok() để thực hiện việc này một cách dễ dàng.

char sign[] = “ ”;

char \*token = strtok(command, sign);

while (token != NULL)

{

args[count++] = token;

token = strtok(command, sign);

}

**d. Thực thi lệnh:**

Hàm thực thi lệnh về cơ bản sẽ có cấu trúc như sau:

pid\_t waitID;

pid\_t childID = fork(); //Tạo tiến trình con

if (childID < 0) //Không tạo được

exit(-1);

else if (childID == 0)

{

//Đoạn code này sẽ được chạy trong tiến trình con

execvp(args[0], args); //Gọi hệ điều hành thực thi

exit(-1); //Thoát tiến trình con nếu execvp() bị lỗi

}

else

{

//Đoạn code này sẽ được chạy trong tiến trình cha

if (concurrentIsNotRequired)

do

{

waitID = wait(&exitCode);

} while (waitID != childID);

}

Hàm fork() tạo tiến trình con sẽ trả về 3 kết quả: nếu bé hơn 0 thì không tạo được tiến trình con, ngược lại, toàn bộ code sẽ được nhân đôi ra, code của tiến trình con sẽ có childID = 0, còn code của tiến trình cha sẽ có childID chính là ID của tiến trình con này.

Bên trong tiến trình con, gọi hàm hệ thống execvp() để hệ điều hành thực thi lệnh. Sau khi hàm này thực thi xong, nó sẽ kết liễu luôn tiến trình con và trả về exitCode = 0. Nếu có lỗi khi execvp() thực thi, tiến trình con sẽ chạy tiếp xuống dưới nên ta phải kết thúc nó bằng lệnh exit(-1).

Trong tiến trình cha, nếu người dùng không nhập & ở cuối câu lệnh thì tiến trình cha phải đợi cho tiến trình con thực hiện xong mới được thoát. Do đó, ta có lệnh kiểm tra concurrent và một vòng lặp while để đợi tiến trình con.

Ngoài việc kiểm tra yêu cầu concurrent ở tiến trình cha, tiến trình con cũng cần kiểm tra lệnh có là redirect hoặc pipe hay không, sau đó mới thực thi cho phù hợp.

**e. IO redirect:**

Nếu người dùng yêu cầu redirect, ta phải xử lý các tham số args trước khi đưa vào execvp(). Các tham số redirect <, >, 2> tương ứng với redirect input, output và error. Ngay sau đó là tên của file được redirect tới.

Mở file này bằng lệnh open() để nhận được file descriptor. Sau đó dùng dup2() để redirect các stdin, stdout hoặc stderr vào file descriptor đó.

//Đoạn code này sẽ được chạy trong tiến trình con

if (strcmp(re, “<”) == 0)

{

fd = open(fileName, O\_RDONLY); //Mở file để đọc

dup2(fd, 0); //Redirect stdin

}

else

{

//Mở file để ghi

fd = open(fileName, O\_WRONLY|O\_CREAT|O\_TRUNC, 0777);

if (strcmp(re, “>”) == 0)

dup2(fd, 1); //Redirect stdout

else

dup2(fd, 2); //Redirect stderr

}

**f. Pipe:**

Pipe là trường hợp thực thi 2 lệnh, mà output của lệnh trước lại là input cho lệnh sau. Vậy lệnh sau sẽ nằm trong tiến trình cha, đợi tiến trình con thực hiện lệnh trước xong, trả về kết quả rồi mới thực hiện.

Để đảm bảo có thể phát triển thêm (vừa pipe vừa redirect vừa concurrent,...), tiến trình con ban đầu sẽ tạo thêm một tiến trình con nữa (tiến trình cháu). Nghĩa là như sau:

childID = fork()

…

else if (childID == 0)

{

//Đoạn code này sẽ được chạy trong tiến trình con

pid\_t = grandID = fork();

…

else if (grandID == 0)

{

//Bên trong tiến trình cháu

…

}

}

Trước tiên, dùng hàm pipe() để tạo ra một ống nối dữ liệu. Ống nối ở đây là 2 file descriptor, hiểu nôm là 2 đầu của ổng, dữ liệu ghi vào đầu này sẽ chảy qua đầu bên kia. Cụ thể, ghi dữ liệu vào fd[1], đọc dữ liệu từ fd[0] với fd là mảng như sau:

int fd[2];

pipe(fd);

Tiến trình cháu sẽ thực hiện lệnh trước, ghi kết quả vào một đầu đường ống. Trong khi đó, tiến trình con sẽ đợi tiến trình cháu chạy xong, rồi lấy input từ đầu đường ống còn lại để thực thi câu lệnh còn lại.

Để làm được điều này, tiến trình cháu sẽ phải redirect output đến fd[1], còn tiến trình con sẽ phải redirect input từ fd[0]. Việc đó hoàn toàn tương tự trong phần IO redirect đã trình bày ở phần d.

**V. Các trang web đã tham khảo**

man7.org/linux

https://linux.die.net/

https://www.geeksforgeeks.com/

--- Hết ---