Báo cáo đồ án 1:

Simple Shell

Môn: Hệ điều hành

Sinh viên thực hiện:

Nguyễn Ngọc Băng Tâm - 1712747

Bùi Thị Cẩm Nhung - 1712645

10/10/2019

1. Tổng quan

1.1. Mô tả đồ án

Shell là một chương trình được phát triển cho phép người dùng sử dụng các dịch vụ của nhân hệ điều hành (Kernal) thông qua một giao diện trung gian.

Mục đích của đồ án này là xây dựng một giao diện shell đơn giản (sử dụng C/C++) hỗ trợ người dùng nhập và thực thi một số câu lệnh hệ thống trong nhiều tiến trình khác nhau.

1.2. Đánh giá mức độ hoàn thành

- Đánh giá tổng thể: 100%
- Chi tiết từng yêu cầu:

Yêu cầu	Hoàn thành	Người thực hiện
Thực thi lệnh trong tiến trình con	x	Nhung
Lưu lịch sử lệnh	x	Tâm
Điều hướng nhập xuất	x	Nhung
Giao tiếp thông qua pipe	x	Tâm

2. Thiết kế

2.1. Phân tích cú pháp lệnh

Hàm: `void parse_command(char *input, char* argv[], int *wait)`

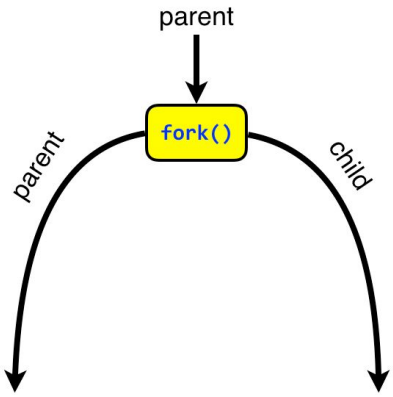
Mô tả:

- Sau khi đọc câu lệnh do người dùng nhập vào, ta thực hiện phân tích cú pháp câu lệnh thành các token. Chẳng hạn người dùng nhập vào "ls -la", token thu được sẽ gồm "ls" và "-la".
- Mỗi token được phân cách nhau bởi ký tự phân cách mặc định là khoảng trắng.
- Token sau khi được phân tách sẽ được lưu vào mảng chứa tham số char* agrv[].
- Thông thường để xác định hai tiến trình sẽ thực hiện song song hay tuần tự, một ký tự '&' sẽ được thêm vào cuối câu lệnh. Một câu lệnh kết thúc với ký tự '&' sẽ được thực thi trong tiến trình con, khi tiến trình con kết thúc thì tiến trình cha mới tiếp tục.
- Ngoài phân tích câu lệnh thành các token, hàm trên còn có tác dụng xác định thứ tự hoạt động của các tiến trình là song song hay tuần tự bằng cách kiểm tra sự xuất hiện của ký tự '&' ở cuối và gán giá trị 0 hoặc 1 tương ứng cho wait.

2.2. Tạo tiến trình mới

Hàm: `int fork()`

Mô tả:



- Trong tiến trình ban đầu, ta gọi lệnh fork() để tạo ra một tiến trình mới. Tiến trình mới này được gọi là tiến trình con (child process) và tiến trình ban đầu được gọi là tiến trình cha (parent process).
- Tiến trình con chính là một bản sao của tiến trình cha. Tuy nhiên tính từ sau khi hàm `fork()` hoàn tất, tiến trình con và tiến trình cha sẽ tách ra thành hai tiến trình độc lập, sở hữu một vùng nhớ riêng.
- Khi thực hiện `fork()` thành công, sẽ có hai giá trị trả về:
    - Tại tiến trình con, giá trị trả về sẽ là 0.
    - Tại tiến trình cha, giá trị trả về sẽ là một số nguyên dương, hay chính là định danh của tiến trình con vừa được tạo (pid). Ta sẽ sử dụng định danh này trong tiến trình cha để theo dõi trạng thái của một tiến trình con.

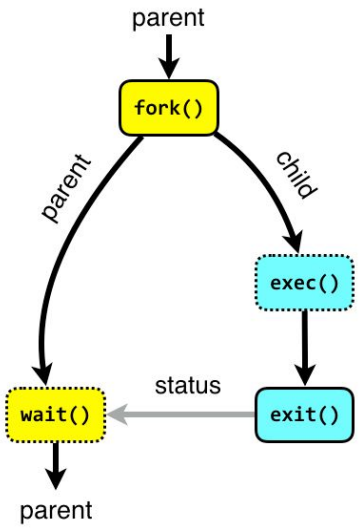
2.3. Thực thi lệnh trong tiến trình con

Hàm: `void child(char* argv[], char* redir_argv[])`

`void parent(pid_t child_pid, int wait)`

Mô tả:

- Tại tiến trình con: Sau khi đã phân tích cú pháp câu lệnh của người dùng và lưu vào mảng tham số argv, ta gọi hàm `execvp(argv[0], argv)` để tiến hành thực thi lệnh.
- Tại tiến trình cha:
  - Nếu giá trị của wait là 1, ta gọi hàm `waitpd(child_pid)` với child_pid là định danh của tiến trình con đã tách ra từ tiến trình cha. Lúc này tiến trình cha sẽ được hoãn cho đến khi tiến trình con thực hiện xong.
  - Ngược lại, hai tiến trình cha và con sẽ thực hiện song song với nhau.



2.4. Phân tích cú pháp lệnh điều hướng nhập xuất

Hàm: `void parse_redirect(char* argv[], char* redir_arg[])`

Mô tả:

- Để điều hướng nhập xuất, trước hết ta cần phải phân tích cú pháp lệnh để biết người dùng đang muốn điều hướng nhập hay xuất và đường dẫn mà họ muốn điều hướng tới là gì.
  - Xác định điều hướng nhập hay xuất được thực hiện bằng cách tìm kiếm ký tự điều hướng trong mảng tham số argv đã được xây dựng trước đó. Trong đó, nếu ký tự điều hướng là '<' sẽ tương đương với việc điều hướng nhập, ký tự điều hướng là '>' tương đương với điều hướng xuất.
  - Đường dẫn điều hướng chính là tham số xuất hiện đằng sau ký tự điều hướng.
- Sao chép ký tự điều hướng và đường dẫn điều hướng vào mảng chứa tham số điều hướng char* redir_argv[]. Chẳng hạn người dùng nhập vào câu lệnh "ls > out.txt", các giá trị trong mảng chứa tham số điều hướng sẽ được tổ chức như sau:

```
redir_argv[0] = ">"
```

```
redir_argv[1] = "out.txt"
```

- Thực hiện xóa các ký tự điều hướng cũng như đường dẫn điều hướng ra khỏi mảng tham số để việc thực thi các câu lệnh được diễn ra bình thường.

2.5. Điều hướng nhập xuất

Hàm: `void child(char* argv[], char* redir_argv[])`

Mô tả:

- Sau khi phân tích câu lệnh điều hướng nhập xuất, ta xác định file descriptor - một số nguyên để định danh file, của đường dẫn điều hướng. Tùy theo loại điều hướng nhập hay xuất mà ta đang muốn thực hiện mà việc xác định file descriptor có thể được thực hiện bằng cách gọi hàm `open()` hay `creat()`.
- Thực hiện thay thế file descriptor của luồn nhập/xuất mặc định bằng file descriptor luồng điều hướng bằng cách gọi hàm `dup2(fd, STDIN_FILENO)` hoặc `dup2(fd, STDOUT_FILENO)` với fd là file descriptor vừa thu được trong bước trên.
- Sau khi điều hướng thành công, đóng file descriptor của luồng điều hướng.

2.6. Tạo chức năng lịch sử

Hàm: `void add_history_feature(char *history[], int &history_count,`

`char* user_input)`

Mô tả:

- Dùng một mảng với kích thước tối đa MAX_HISTORY để lưu các lệnh đã từng được nhập vào. Thêm vào đến khi mảng đầy, giải phóng lệnh đầu tiên và lưu phần tử mới vào cuối mảng. Việc thêm lệnh vào mảng được thực hiện bằng hàm `strcpy(char* des, char* src)`.
- Khi người dùng nhập vào chuỗi "!!", in ra màn hình lệnh gần nhất và thực thi lệnh đó. Trong trường hợp lịch sử trống (history_count = 0), báo lỗi "No commands in history".

2.7. Kiểm tra lệnh có chứa ký tự pipe

Hàm: `bool parse_pipe(char* argv[], char *child01_argv[], char *child02_argv[])`

Mô tả:

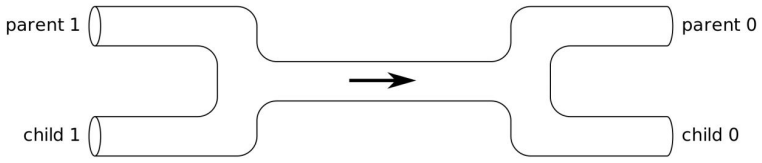
- Hàm nhận vào chuỗi các token của lệnh sau khi đã tách khoảng trắng. Duyệt qua từng token và tìm vị trí của ký tự '|'. Do đề bài yêu cầu chỉ xét các lệnh chứa 1 ký tự '|' và không đi kèm toán tử điều hướng nên kết quả trả về chỉ cần là có ký tự '|' hay không.
- Khi tìm được ký tự '|', lưu các token trước vị trí của '|' vào chuỗi `child01_argv`, các token sau '|' lưu vào chuỗi `child02_argv`. Kết thúc mỗi chuỗi đều là NULL.

2.8. Thực thi lệnh khi có ký tự '|'

Hàm: void `exec_with_pipe(char* child01_argv[], char* child02_argv[])`

Mô tả:

- Hàm nhận vào nhóm lệnh trước và sau ký tự '|'.
- Cần tạo 1 đường ống cho phép giao tiếp giữa hai tiến trình, gồm 2 con trỏ file descriptor (1 cho input và 1 cho output).



- Để đường ống làm việc hiệu quả, luôn phải đóng phần cuối của đường ống không cần dùng. Ví dụ, tiến trình 1 truyền dữ liệu cho tiến trình 2, nên tiến trình 1 cần đóng pipefd[0] (đóng tính năng đọc), và tiến trình 2 cần đóng pipefd[1] (đóng tính năng ghi). Khi việc đọc/ ghi tiến trình hoàn thành, cần đóng file descriptor liên quan.
- Sau khi khởi tạo đường ống, dùng hàm `fork()` để khởi tạo 2 tiến trình con. Do output của tiến trình con 1 sẽ trở thành input của tiến trình con 2, dùng hàm `dup2(pipefd[1], STDOUT_FILENO)` để ghi output vào stdout và đóng pipefd[0]. Ngược lại với tiến trình 2, dùng hàm `dup2(pipefd[0], STDIN_FILENO)` đọc input từ stdin và đóng pipefd[1].
- Tiến trình 1 thực hiện lệnh dựa trên mảng tham số child01_argv, tiến trình 2 thực hiện lệnh dựa trên mảng tham số child02_argv.

3. Tài liệu tham khảo

1. ITC Lecture Note - Operating System 2018, Process Management
2. /linux/man-pages/man2
3. Create pipes in C
4. CS 702 Spring 2005
5. CSCI3150