**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**٭٭٭٭٭**

****

**BÁO CÁO ĐỒ ÁN: SIMPLE SHELL**

**MÔN HỌC: HỆ ĐIỀU HÀNH**

LỚP : 18CNTN

SINH VIÊN THỰC HIỆN : 18120019 – Nguyễn Hoàng Dũng

18120052 – Lê Hạnh Linh

*Thành phố Hồ Chí Minh, ngày 5 tháng 11 năm 2020*

Mục Lục

[**I.** **BIẾN TOÀN CỤC** 2](#_Toc55686294)

[**a.** **int pipes[]:** 2](#_Toc55686295)

[**b.** **int user\_read, user\_write:** 2](#_Toc55686296)

[**II.** **ĐỊNH NGHĨA HÀM** 2](#_Toc55686297)

[**a.** **void io\_redirect(char sign, char \*file)** 2](#_Toc55686298)

[**b.** **void parse\_iostream(char \*\*agrv, bool \*background)** 2](#_Toc55686299)

[**c.** **void parse\_command(char \*args, char \*\*agrv)** 3](#_Toc55686300)

[**d.** **pid\_t execute\_command(char \*\*argv, int pipe\_read, int pipe\_write)** 3](#_Toc55686301)

[**e.** **void count\_command(char \*str, char \*\*cmd)** 3](#_Toc55686302)

[**f.** **void pipe\_command(int index, int \*pipe\_read, int \*pipe\_write)** 4](#_Toc55686303)

[**g.** **bool is\_internal\_command(char \*\*argv)** 4](#_Toc55686304)

[**h.** **void main\_process(char \*line)** 4](#_Toc55686305)

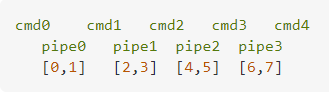
[**i.** **bool check\_last\_command(char \*line, char \*hist)** 5](#_Toc55686306)

[**j.** **void open\_pipes()** 5](#_Toc55686307)

[**III.** **PHÂN CÔNG CÀI ĐẶT** 5](#_Toc55686308)

[**IV.** **DEMO** 6](#_Toc55686309)

1. **BIẾN TOÀN CỤC**
   1. **int pipes[]:** 
      * Mảng chứa lần lượt các giá tri read end và write end của các pipe.
      * Mỗi pipe chiếm 2 vị trí liên tiếp: chẵn là read end, lẻ là write end.



* 1. **int user\_read, user\_write:** 
     + Lưu giá trị file descriptor để read và write với user.
     + Gọi hàm dup() để lấy giá trị mặc định từ STDIN/OUT.

1. **ĐỊNH NGHĨA HÀM**
   1. **void io\_redirect(char sign, char \*file)**

Input:

* + - sign: kí tự < hoặc >
    - file: chuỗi kí tự chứa tên file

Xử lý:

* + - Nếu sign là < thì mở file theo chế độ đọc và gán cho user\_read.
    - Nếu sign là > thì mở file theo chế độ ghi và gán cho user\_write.
  1. **void parse\_iostream(char \*\*agrv, bool \*background)**

Input:

* + - argv: biến trỏ đến danh sách các con trỏ quản lý argument)
    - background: flag chứa tín hiệu đánh dấu parent chờ child process

Xử lý:

* + - Duyệt argv nếu có các kí tự <, > thì gọi hàm io\_redirect.
    - Nếu có & thì gán background = true
      * Báo hiệu parent chờ child process kết thúc
  1. **void parse\_command(char \*args, char \*\*agrv)**

Input:

* + - args: chuỗi kí tự chứa 1 command user nhập vào
    - argv: biến để chứa danh sách các con trỏ quản lý argument

Xử lý:

* + - Từ args, tách thành các argument và lưu vào argv
  1. **pid\_t execute\_command(char \*\*argv, int pipe\_read, int pipe\_write)**

Input:

* + - argv: biến để chứa danh sách các con trỏ quản lý argument
    - pipe\_read: read end của pipe kề trước để child process đọc input
    - pipe\_write: write end của pipe kề sau để child process ghi output

Xử lý:

1. Gọi hàm fork() khởi tạo child process.
2. Nếu là child process:
   * + Gọi dup2() để đọc ghi từ pipe\_read và pipe\_write.
     + Gọi hàm execvp() và truyền argv để thực hiện command.
3. Nếu là parent process:
   * + Close file descriptor (pipe\_read, pipe\_write)
     + Chờ child process nếu có background = true

Ouput:

* + - Child process ID
  1. **void count\_command(char \*str, char \*\*cmd)**

Input:

* + - str: chứa chuỗi command user nhập (có thể có pipe)
    - cmd: biến để chứa danh sách các con trỏ quản lý commad.

Xử lý:

* + - Đếm số lượng command.
    - Lưu con trỏ đến các command này vào 1 mảng con trỏ do cmd quản lý.
  1. **void pipe\_command(int index, int \*pipe\_read, int \*pipe\_write)**

Input:

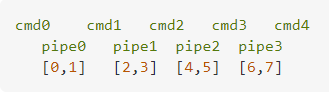
* + - index: thứ tự của command hiện tại
    - pipe\_read: read end của pipe kề trước để child process đọc input
    - pipe\_write: pipe\_write: write end của pipe tương ứng để child process ghi output

Xử lý:

* + - Nếu là command đầu: pipe\_read = user\_read
    - Nếu là command cuối: pipe\_write = user\_write
    - Command thứ i sẽ:

Đọc input (pipe\_read) từ pipes[i \* 2 – 2]

Ghi output (pipe\_write) ra pipes[i \* 2 + 1]



* 1. **bool is\_internal\_command(char \*\*argv)**

Input:

* + - argv: biến để chứa danh sách các con trỏ quản lý argument

Xử lý:

* + - Xét xem argv lưu command của shell (cd, exit, pwd) hay không.

Output:

* + - True: command đó là internal
    - False: không phải là internal
  1. **void main\_process(char \*line)**

Input:

* + - line: chuỗi kí tự user nhập

Xử lý:

* + - Gọi các hàm trên để xử lý lần lượt các command.
  1. **bool check\_last\_command(char \*line, char \*hist)**

Input:

* + - line: chuỗi kí tự user nhập
    - hist: chuỗi kí tự trước đó user đã nhập

Xử lý:

* + - Xét input của user là command hay là yêu cầu thực hiện command gần nhất trong history (!!).
    - Nếu là command thì copy vào buffer hist. Ngược lại thực hiện command trong buffer hist.

Output:

* + - True: Nếu command là !! và trước đó có command, hoặc command khác !!
    - False: Nếu command là !! và trước đó không có command
  1. **void open\_pipes()**

Global:

* + - pipes[]: mảng chứa lần lượt các giá tri read end và write end của các pipe

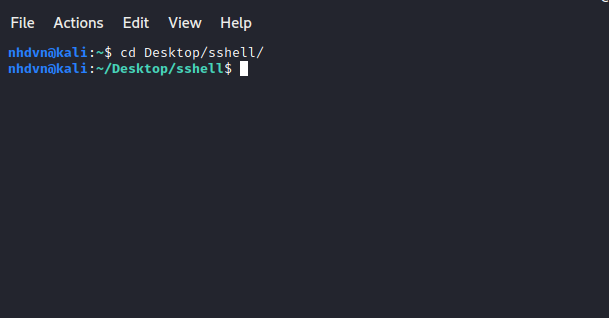
Xử lý:

* + - Gọi hàm pipe() tại các vị trí pipes + i \* 2

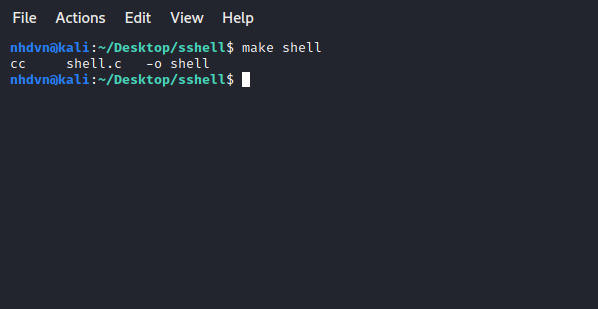
1. **PHÂN CÔNG CÀI ĐẶT**

|  |  |  |
| --- | --- | --- |
| **Hàm** | **Người phụ trách** | **Mức độ hoàn thành** |
| io\_redirect | Lê Hạnh Linh | 100% |
| parse\_iostream | Nguyễn Hoàng Dũng | 100% |
| parse\_command | Nguyễn Hoàng Dũng | 100% |
| execute\_command | Nguyễn Hoàng Dũng | 100% |
| count\_command | Lê Hạnh Linh | 100% |
| pipe\_command | Lê Hạnh Linh | 100% |
| main\_process | Lê Hạnh Linh | 100% |
| is\_internal\_command | Nguyễn Hoàng Dũng | 100% |

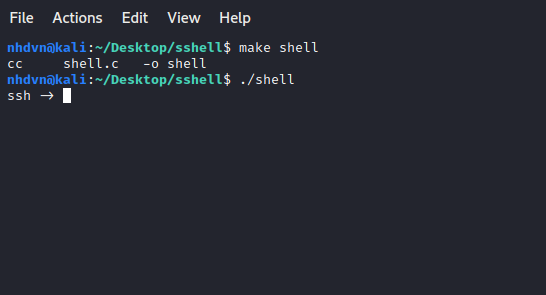
1. **DEMO**
2. Change dir to project directory



1. Compile shell.c into executable binary file



1. Run the binary



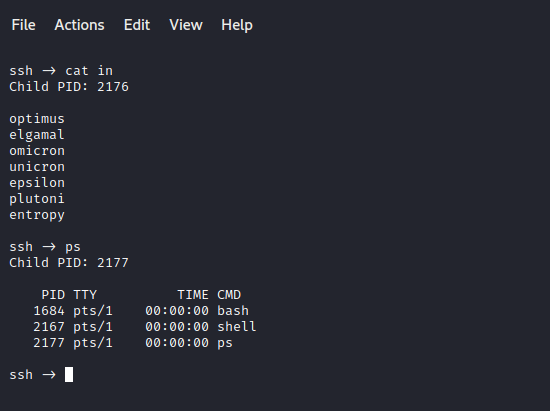
1. Simple command with child process

Cách kiểm tra:

* + - In ra pid của child process sau khi gọi hàm fork()
    - Gọi command với lệnh ps

Giải thích:

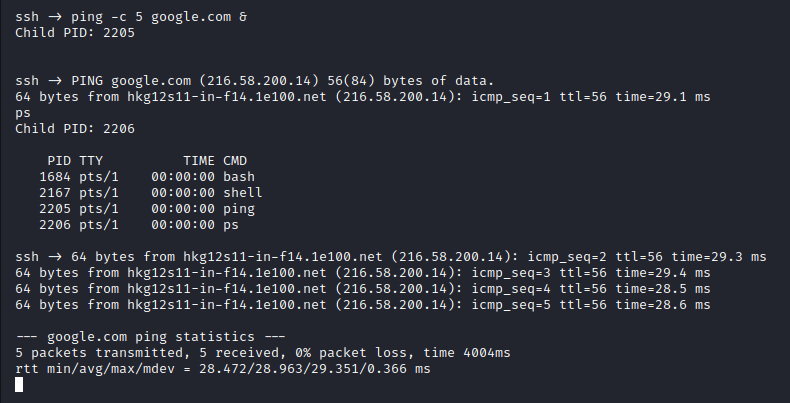
Từ parent, ta fork child process để thực hiện lệnh ps và in ra pid của child process. Mà bản thân lệnh ps in ra các process status đang chạy.



1. Simple command with &

Cách kiểm tra:

* + - Gọi lệnh ping để và thêm & để cho child process chạy background.
    - Gõ tiếp lệnh ps sẽ thấy được thực hiện ngay, trong khi đó lệnh ping vẫn tiếp tục in ra màn hình.
    - Lệnh ps in đồng thời cả pid của child process thực hiện lệnh ping và child proces thực hiện lệnh ps.

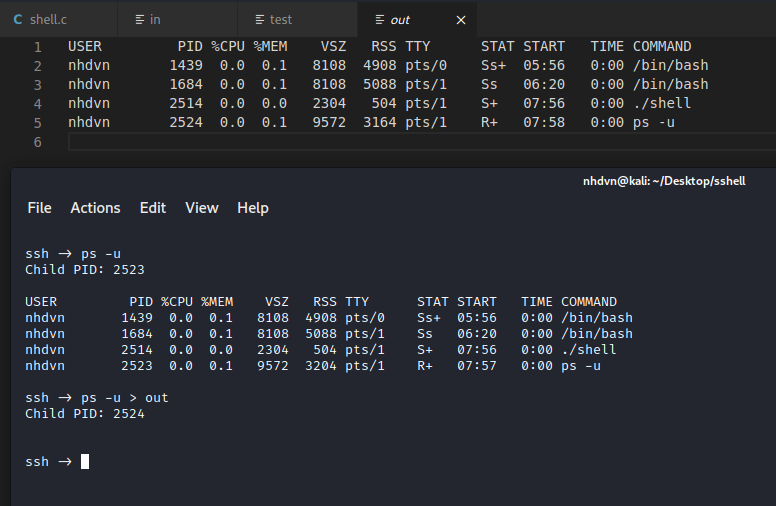


Giải thích:

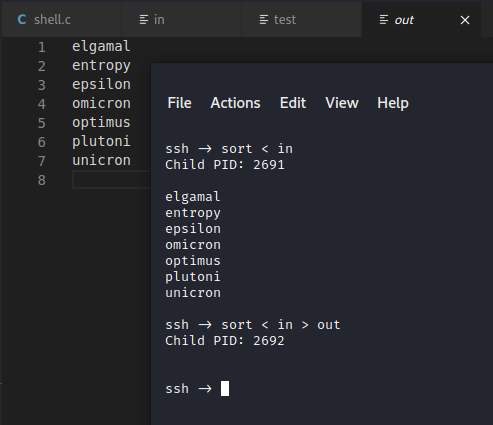
* + - Nếu không có & thì lệnh ps phải đợi lệnh ping thực hiện hết trước.
    - Lúc này, lệnh ps được thực hiện thì không còn thấy lệnh ping trong process status nữa.



1. Ouput redirection



1. Input redirection



1. Pipe multiple command

