ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH  
  
ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN  
  
MÔN HỆ ĐIỀU HÀNH

GVHD: Trần Trung Dũng

BÁO CÁO ĐỒ ÁN 1

Nhóm:  
Lê Thanh Trường Vinh. MSSV: 1712910.  
Phan Nhật Vinh. MSSV: 1712914.

Nội dung

[I Phân Chia: 2](#_Toc21561769)

[II Thiết kế đồ án: 2](#_Toc21561770)

[1 Sử dụng fork(): 3](#_Toc21561771)

[2 Sử dụng exec(): 4](#_Toc21561772)

[3 Tính năng lưu lại lịch sử: 6](#_Toc21561773)

[4. Tái định hướng Input và Output: 6](#_Toc21561774)

[5. Giao tiếp qua Pipe: 7](#_Toc21561775)

[III Các Hàm/ Lời Gọi Hệ Thống Sử Dụng: 9](#_Toc21561776)

[1 Fork() (syscall): 9](#_Toc21561777)

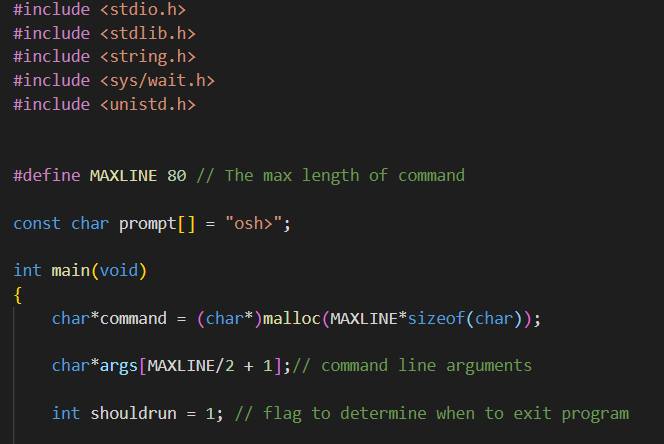
[2 Execvp() (syscall): 9](#_Toc21561778)

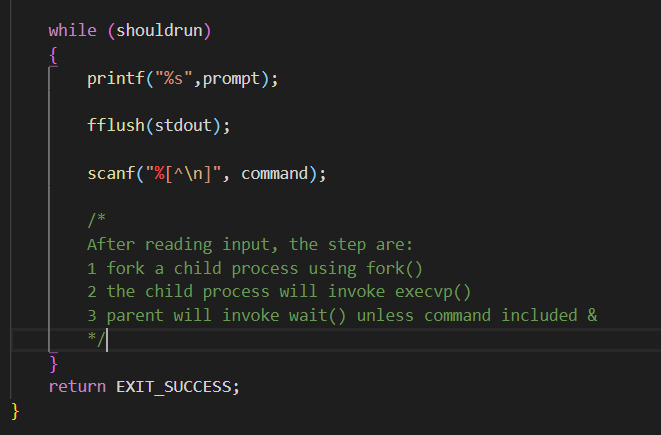
[3 Open() (syscall): 9](#_Toc21561779)

[Return value: file descriptor. 9](#_Toc21561780)

[4 dup2() (function): 9](#_Toc21561781)

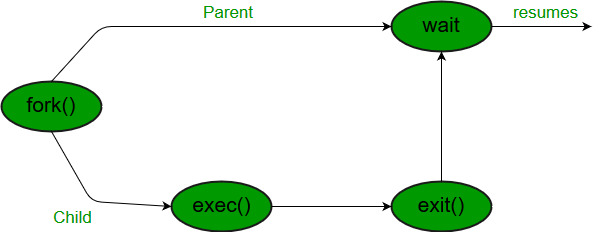
# I Thiết kế đồ án:

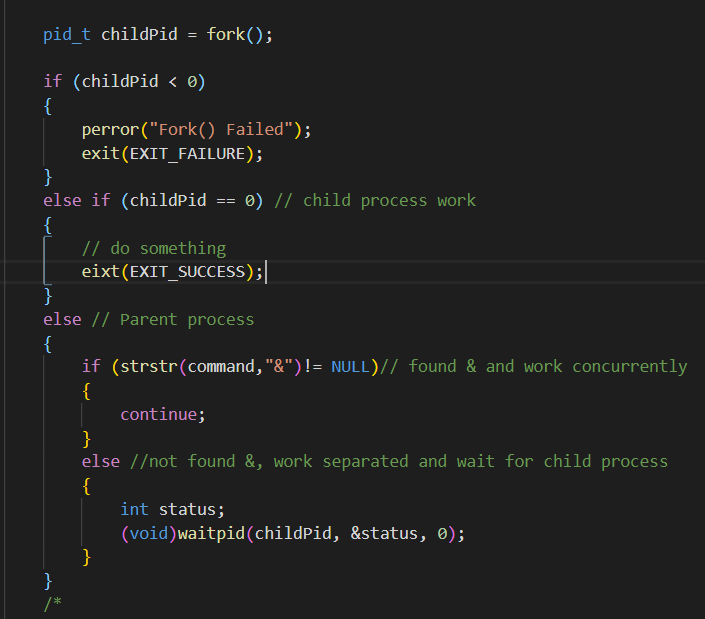
Giao diện của Simple Shell luôn hiển thị một prompt, sau đó người dùng sẽ nhập một câu lệnh nào đó. Trong đồ án này, prompt sẽ là osh>. Trong việc thực hiện tiến trình, một tiến trình (gọi là Parent) sẽ đọc câu lệnh từ người dùng, và tạo ra một tiến trình khác (gọi là Child) để xử lý câu lệnh đó.Thông thường, tiến trình Parent sẽ đợi tiến trình Child hoàn thành rồi mới tiếp tục. Tuy nhiên, UNIX Shell cũng cho phép tiến trình Child có thể chạy nền, hoặc là chạy song song với tiến trình Parent. Để thực hiện được điều đó, chúng ta sẽ thêm dấu & vào cuối mỗi câu lệnh.



Để thực hiện được việc tạo tiến trình Child chúng ta cần phải sử dụng lời gọi hệ thống (systemcall) fork().

## 1 Sử dụng fork():

 Trong máy tính đặc biệt là các hệ thống sử dụng UNIX, fork() là một hoạt động mà một tiến trình có thể tạo ra một bản sao của chính nó. Đây chính là một lời gọi hệ thống, được thực hiện bên trong kernel. Fork() là một phương thức chính để tạo các tiến trình trong các hệ thống tương tự UNIX.



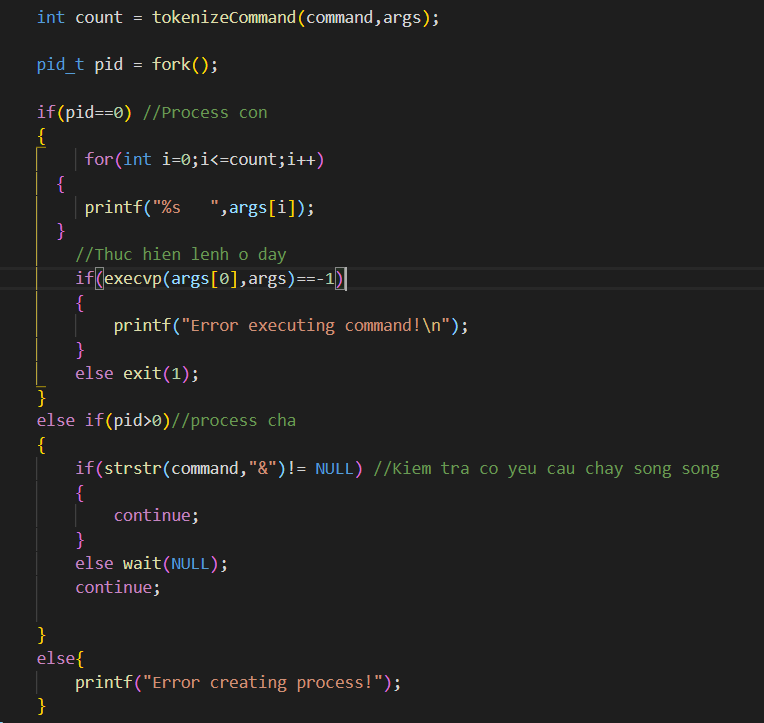
Fork() work with &.

Quy trình của lời gọi hàm fork()

## 2 Sử dụng exec():

Để xử lý được câu lệnh, chúng ta cần sử dụng hàm exec() ( cụ thể sẽ sử dụng hàm execvp(char\* command, char\* params[]) ).

Để có thể sử dụng được execvp, đầu tiên chúng ta cần phải tách câu lệnh command thành từng thành phần nhỏ hơn, và được lưu trong mảng args, ví dụ command là ps -ael thì những giá trị được lưu trong mảng args là: args[0] = “ps”, args[1] = “-ael”, args[2] = NULL. Và câu lệnh trên sẽ được thực thi theo hàm execvp(args[0],args). Và chúng ta cần phải kiểm tra người dùng có nhập & trong lệnh hay không, để xác định rằng tiến trình Parent có cần đợi tiến trình Child để thoát hay không.



Execute command.

Parse command into token for args array.

## 3 Tính năng lưu lại lịch sử:

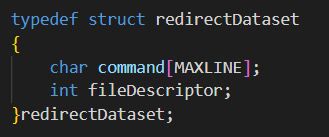
Simple Shell cung cấp tình năng lịch sử cho phép người dùng có thể thực thi lại câu lệnh gần nhất bằng việc nhập vào !!. Nếu chưa có lịch sử nào (lần đầu tiên sử dụng Shell) thì khi nhập !! sẽ hiển thị “No commands in history”.

Để thực hiện tính năng này, shell sẽ tạo ra một file để chứa lịch sử (ví dụ ở đây sử dụng file tên history.txt). Mỗi lần chạy, Shell sẽ load dữ liệu từ file history.txt lên một chuỗi char\* HistoryCommand. Nếu câu lệnh đó có thể thực thi được thì ngay lập tức lưu lại câu lệnh đó vào file history.txt.

## 4. Tái định hướng Input và Output:

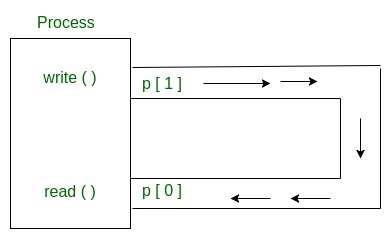
Shell cần hỗ trợ người dùng toán tử tái định hướng “>” và “<”. Toán tử “>” dùng để lưu output của một command nào đó vào file, và toán tử “<” dùng để input một command từ một file. Ví dụ: osh>ls > output.txt thì output của command ls sẽ được lưu vào file output.txt, hay osh>sort < input.txt thì file input.txt sẽ được xem như là input cho command sort.

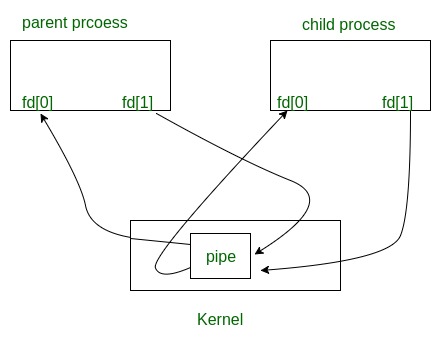
Để thực hiện các redirection input và output cần sử dụng hàm dup2(), hàm này sẽ tạo bản sao của một file descriptions đang tồn tại sang một file description khác.

Để thực hiện hàm này một cách dễ dàng, chúng ta sẽ tạo một kiểu dữ liệu để lưu lại command và fileDescriptor, với fileDescriptor dùng để truy cập vào input/output, nhận 3 giá trị là 0, 1 hoặc 2.

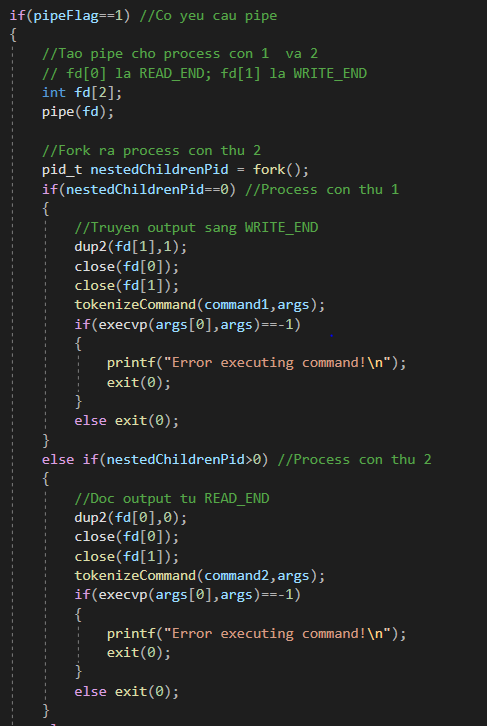
## 5. Giao tiếp qua Pipe:

Một cách đơn giản nhất, Pipe là sự kết nối giữa 2 tiến trình, output của một tiến trình sẽ là input của tiến trình còn lại. Hàm pipe() trong UNIX sẽ tạo ra một cặp file descriptor trỏ tới pipe inode, và đặt chúng vào trong một mảng fd[2], với fd[0] cho việc đọc và fd[1] cho việc ghi.

Trong Shell, Pipe cho phép output của một command sẽ là input của một command khác. Cả hai command sẽ thực thi riêng biệt nhau và sẽ sẽ giao tiếp với nhau thông qua hàm pipe() của UNIX. Để thực hiện việc này một cách đơn giản nhất, tiến trình Parent sẽ tạo ra một tiến trình Child, và tiến trình Child này cũng sẽ tạo ra một tiến trình Child khác và thành lập pipe giữa nó và tiến trình Child mà nó tạo ra. Việc thiết lập hàm pipe() cũng cần sử dụng hàm dup2() để lưu lại section trước đó.



Pipe()



# II Các Hàm/ Lời Gọi Hệ Thống Sử Dụng:

## 1 Fork() (syscall):

#include <unistd.h>

Prototype: fork().

Return value:

* Negative number: khởi tạo tiến trình con lỗi.
* Zero value: tạo thành công tiến trình con.
* Positive value: trả về tiến trình cha.

## 2 Execvp() (syscall):

#include <unistd.h>

Prototype: execvp(char\*path, char\* const agrs[]).

Return value: không có trả về giá trị, nhưng nếu trả về -1 thể hiện việc thực thi lỗi.

## 3 Open() (syscall):

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

Prototype: open(const char\*pathname, int flags, mode\_t mode).

Return value: file descriptor.

## 4 dup2() (function):

#include <unistd.h>

Prototype: dup2(int oldfd, int newfd).

Return value: Trả về một new descriptor, hoặc -1 nếu có lỗi xảy ra.