



Báo cáo đồ án 1

SIMPLE SHELL

Môn: Hệ điều hành

Sinh viên thực hiện

Trần Quốc Bảo – 18120111

Phạm Trọng Long – 18120135

Lê Minh Khoa – 18120415

Mục lục

1. Tổng quan.....	3
1.1. Mô tả đồ án.....	3
1.2. Đánh giá mức độ hoàn thành.....	3
1.3. Mức độ đóng góp.....	4
2. Thiết kế chương trình.....	4
2.1. Phân tích cú pháp lệnh	4
2.2. Tạo tiến trình con	4
2.3. Thực hiện lệnh trong tiến trình con	6
2.4. Lưu trữ lịch sử các lệnh:	7
2.5. Điều hướng nhập xuất	7
2.6. Tổ chức giao tiếp giữa tiến trình cha và con thông qua pipe	7
2.6.1. Phân tích cú pháp lệnh pipe.....	7
2.6.2. Thực hiện lệnh pipe	8
2.7. Công dụng của một số system call:	9
3. Kiểm thử(screen shot).....	9
3.1. Simple commands with child processes.....	10
3.2. Simple command with '&'	10
3.3. Input redirection	11
3.4. Output redirection	11
3.5. Two commands with a pipe.....	12
3.6. Providing a history feature	12
4. Tài liệu tham khảo.....	12

1. Tổng quan

1.1. Mô tả đồ án

Shell là chương trình giúp người dùng giao tiếp với hệ thống Unix bằng cách đọc và thực thi thông tin được nhập từ bàn phím.

Đồ án Simple Shell có mục đích thiết kế một chương trình Shell bằng ngôn ngữ C với các chức năng cơ bản như đọc, thực thi lệnh trong các tiến trình khác nhau, điều hướng nhập xuất, lưu lịch sử.

1.2. Đánh giá mức độ hoàn thành

Mức độ hoàn thành tổng thể đồ án: 100%

Mức độ hoàn thành từng yêu cầu:

Yêu cầu	Mức độ hoàn thành	Người thực hiện
Tạo và thực thi lệnh trong tiến trình con	100%	Long
Lưu lịch sử thực hiện lệnh	100%	Khoa
Điều hướng nhập và xuất	100%	Bảo Long
Các tiến trình giao tiếp thông qua pipe	100%	Khoa Bảo

1.3. Mức độ đóng góp

Thành viên	Mức độ đóng góp(%)	Kí tên
Trần Quốc Bảo	33.(33)	Bảo
Phạm Trọng Long	33.(33)	Long
Lê Minh Khoa	33.(33)	Khoa

2.Thiết kế chương trình

2.1. Phân tích cú pháp lệnh

Hàm:

```
void parseArgsCommand(char *str, char* args[], bool& backgroundCase)
```

Mô tả: Hàm `parseArgsCommand` có chức năng tách từ chuỗi input nhập từ bàn phím và kiểm tra trường hợp chạy song song tiến trình:

Khi được gọi, đầu tiên hàm sẽ xóa dữ liệu trong mảng `args`.

Sau đó kiểm tra xem ký tự cuối cùng được nhập vào có phải là "&" hay không. Nếu có thì gán biến `backgroundCase` bằng 1, ngắt chuỗi nhập vào tại vị trí đó, ngược lại bằng 0.

Sau đó dùng hàm `strtok` trong thư viện `<string.h>` nhằm tách các mảng ký tự giữa các dấu cách.

2.2. Tạo tiến trình con

Hàm: `pid_t fork()`

Mô tả:

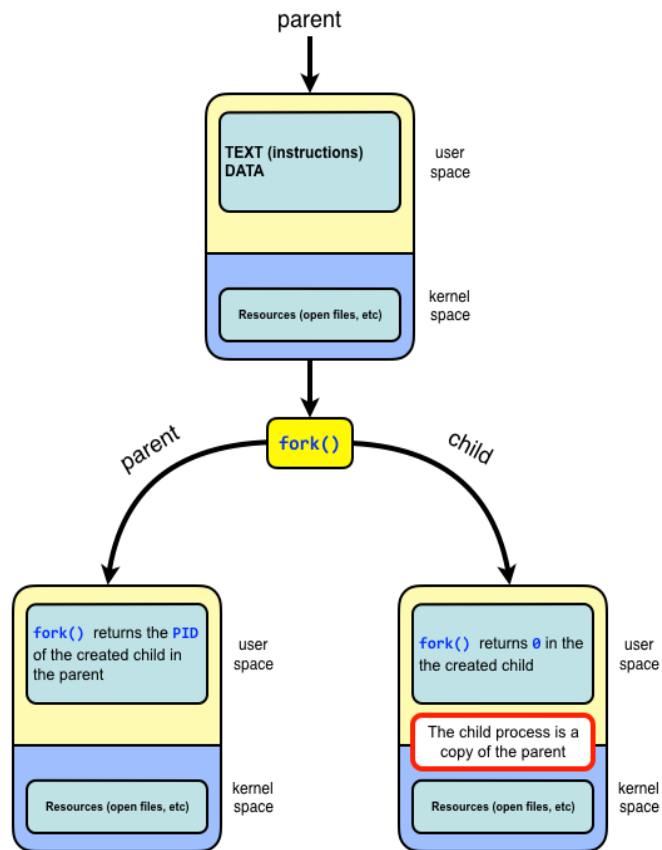
Khi ta gọi hàm `fork()`, tạo ra bản sao của tiến trình đang gọi, bản sao này gọi là tiến trình con.

Memory cũng được sao chép thành 1 bản mới, và vùng nhớ của 2 tiến trình là độc lập nên những thay đổi trên vùng nhớ của tiến trình con sẽ không ảnh hưởng lên vùng nhớ của tiến trình cha và ngược lại.

Trước khi hàm `fork()` thực hiện xong và trả về giá trị, tiến trình con đã được nhân bản, và giá trị trả về của hàm `fork()` ở tiến trình cha và tiến trình con là khác nhau.

Giá trị trả về của hàm `fork` là `pid_t` (thực chất là 1 số `int`):

- Trả về giá trị 0 khi đang ở tiến trình con
- Trả về `id` của tiến trình con vừa tạo khi đang ở tiến trình cha.



2.3. Thực hiện lệnh trong tiến trình con

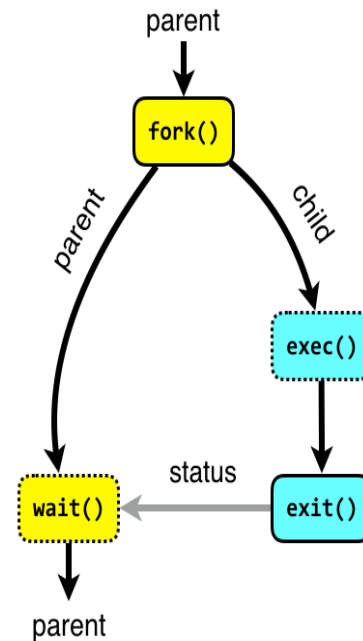
Hàm:

```
void runCommand(char* args[], char* redirect, int redirectFlag)
```

Mô tả: Để tiến hành tạo ra tiến trình con và thực thi lệnh, ta làm như sau:

Đầu tiên, ta gọi hàm `fork()` để tạo ra tiến trình con. Sau đó ta kiểm tra giá trị `pid_t` được trả về của hàm `fork()`.

- Nếu giá trị âm, tức hàm `fork()` đã thất bại trong việc tạo tiến trình con.
- Nếu giá trị là 0, tức ta đang ở tiến trình con, ta sẽ xử lý các tham số truyền vào, sử dụng hàm `execvp(char *command, char *params[])` để thực thi lệnh.
- Nếu giá trị là 1 số nguyên dương, ta đang ở tiến trình cha, ta sẽ tùy vào tham số có kí tự "&" ở cuối hay không. Nếu không có kí tự "&" thì dừng lại chờ tiến trình con hoàn thành, ngược lại tiếp tục tiến trình cha.



Bức ảnh bên mô tả khái quát về quá trình thực thi lệnh ở tiến trình con. Sau khi khởi tạo tiến trình con thành công, tiến trình con thực thi lệnh và ngưng khi hoàn thành. Trong khi đó, tiến trình cha đợi tiến trình con bằng hàm `wait()` / `waitpid()`.

Hàm `runCommand()` được gọi bởi tiến trình con khi không sử dụng pipe:

- Nếu cờ `redirectFlag` được bật (1 nếu là input, 2 nếu là output), ta sẽ thực hiện điều hướng nhập xuất.
- Đầu tiên ta `open`, mở file, nếu file không tồn tại trong trường hợp output, ta có thể tạo file bằng `creat()`.
- Sau đó, ta thay thế luồng nhập/ xuất của tiến trình bằng cách dùng hàm `dup2()`.
- Sau khi điều hướng thành công, ta đóng file lại.
- Cuối cùng ta gọi hàm `execvp()` để thực thi lệnh.

2.4. Lưu trữ lịch sử các lệnh:

Hàm:

```
void historyProceed(int& historyCount, char* history[], char inputStr[])
```

Mô tả:

Dùng mảng `char* history[]` lưu những câu lệnh đã được nhập với số phần tử nhiều nhất là `MAX_HISTORY`. Nếu vượt quá giới hạn thì ta loại bỏ câu lệnh đầu tiên được lưu, sau đó lưu câu lệnh mới vào cuối mảng (mảng này hoạt động theo cơ chế FILO).

Khi người dùng nhập vào lệnh “!”, tiến hành in ra màn hình câu lệnh gần nhất và thực thi câu lệnh này.

2.5. Điều hướng nhập xuất

Hàm: `int parseRedirect(char* args[], char*& redirect)`

Mô tả:

Để thực hiện điều hướng nhập xuất, ta cần phải xác định cú pháp người dùng đã nhập là gì và đường dẫn mà người dùng muốn điều hướng tới.

Xác định tham số điều hướng bằng cách tìm kiếm trong mảng `args[]` đã được xây dựng từ câu lệnh của người dùng. Nếu kí tự tìm được là ‘<’ tương ứng với điều hướng nhập và kết quả hàm sẽ trả về 1, ‘>’ tương ứng với điều hướng xuất và hàm trả về 2, nếu chỉ là lệnh bình thường không điều hướng thì hàm sẽ trả về giá trị 0.

Đường dẫn người dùng muốn điều hướng tới sẽ được sao chép và lưu ở mảng chứa đường dẫn điều hướng `char* redirect`.

Ví dụ: nếu người dùng nhập `osh > sort < input.txt` thì hàm trên sẽ trả về 1, `redirect = "input.txt"`.

2.6. Tổ chức giao tiếp giữa tiến trình cha và con thông qua pipe

2.6.1. Phân tích cú pháp lệnh pipe

Hàm: `bool parsePipe(char* args[], char* argsPipe[2][MAX_ARG_SIZE])`

Mô tả:

Trước hết, ta cần xác định từng thành phần của lệnh và tách riêng chúng ra. Do đề bài chỉ giới hạn nhiều nhất là một kí tự "|" nên ta chỉ cần xác định xem kí tự "|" nằm ở vị trí nào. Nếu như có xuất hiện kí tự "|", hàm sẽ trả về True, ngược lại hàm trả về False.

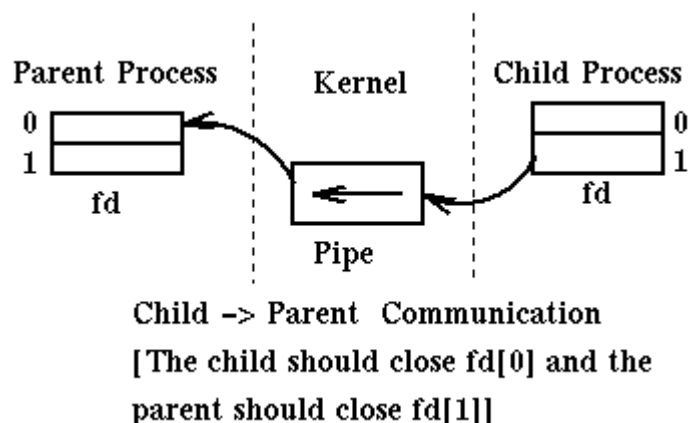
Sau khi tìm được vị trí của token đánh dấu sử dụng pipe ("|"), ta lưu tất cả những token trước vị trí này vào `argsPipe[0]`, và tất cả những token sau vị trí này vào `argsPipe[1]`.

2.6.2. Thực hiện lệnh pipe

Hàm: `void runPipedCommand(char* argsPipe[2][MAX_ARG_SIZE])`

Mô tả:

- Hàm nhận vào tham số là những token trước kí tự "|" (`argsPipe[0]`), và sau kí tự "|" (`argsPipe[1]`).
- Hàm tạo ra một Pipe dùng để liên lạc giữa 2 tiến trình, về cơ bản đường ống này tạo ra 2 description file, một đầu ống có tác dụng ghi (output) và đầu còn lại có tác dụng đọc (input).
- Nguyên tắc của việc sử dụng pipe hiệu quả và tránh xảy ra lỗi là phải luôn đóng một phần pipe không dùng đến. Ví dụ như hình dưới đây: khi ta sử dụng tiến trình con để ghi, thì tiến trình con cần phải đóng tính năng đọc (`pipefd[0]`) và tiến trình cha cần đóng tính năng ghi (`pipefd[1]`). Tiến trình con và tiến trình cha thực thi lệnh dựa trên 2 tham số `argsPipe[0]` và `argsPipe[1]`.



Các bước tiến hành:

- Sau khi khởi tạo thành công pipe, ta dùng hàm `fork()` để lần lượt sinh ra 2 tiến trình con.
- Output do tiến trình con 1 sinh ra sẽ được dùng làm input của tiến trình con 2, do đó để thuận tiện cho việc xử lí, ta chuyển luồng ghi output này vào luồng

`stdout` bằng lệnh `dup2(pipefds[1], fileno(stdout))` và thực hiện đóng tính năng đọc (`pipefd[0]`).

- Tương tự ta cũng chuyển luồng đọc input của tiến trình 2 vào luồng `stdin` bằng lệnh `dup2(pipefds[0], fileno(stdin))` và thực hiện đóng tính năng ghi (`pipefd[1]`).

2.7. Công dụng của một số system call:

execvp: Tạo một tiến trình nhỏ chạy độc lập với tiến trình hiện tại giúp thực thi các lệnh trong hệ thống. Hàm `execvp` có hai tham số: `int argc`, `char **argv`. Khi hàm được gọi, lệnh được lưu ở tham số `argc` sẽ được gọi với các tham số của hàm là `argv`.

wait(): Hàm `wait()` được gọi nhằm mục đích đợi tiến trình con bị ngắt, bị dừng hoặc được tiếp tục. Sau khi đợi xong, tiến trình cha tiếp tục thực thi các câu lệnh khác như bình thường.

open: Dùng để mở tập tin với đường dẫn là tham số đầu tiên của hàm. Hàm trả về một biến `int` nhỏ nhất có thể, không âm là tham số mô tả của tập tin.

dup2, dup:

- `int dup(int oldfd)`: Tạo một bản sao chép của biến mô tả file `oldfd`, trả về giá trị số nguyên nhỏ nhất có thể là giá trị của biến mô tả file mới.
- `int dup2(int oldfd, int newfd)`: thực hiện công việc giống như hàm `dup`, nhưng thay vì dùng số nguyên nhỏ nhất có thể thì ta dùng giá trị của biến `newfd`.

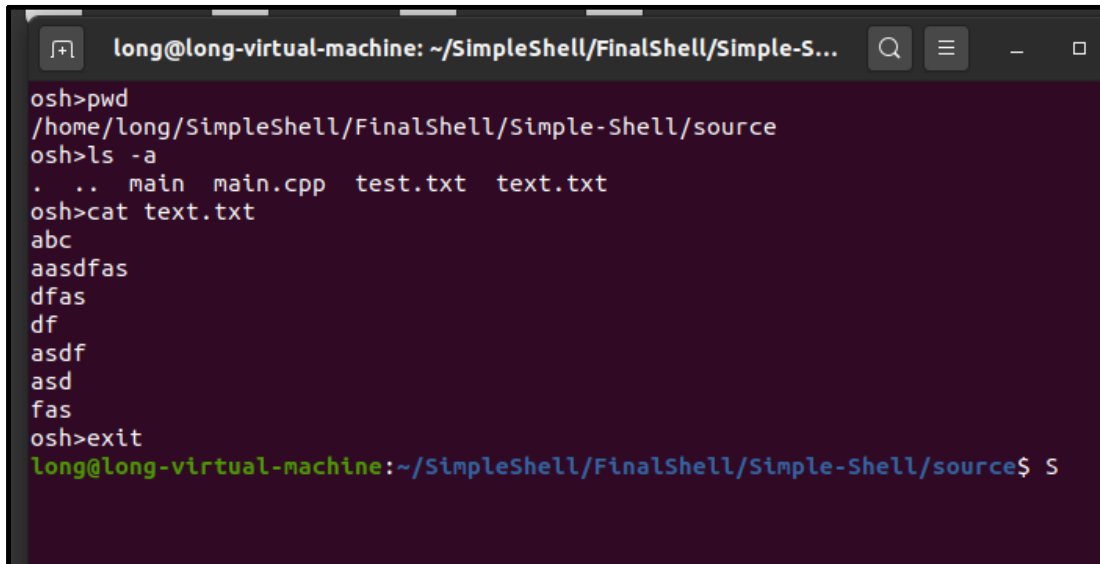
3. Kiểm thử(screen shot)

Cách chạy chương trình:

- `g++ main.cpp -o program`
- `./program`

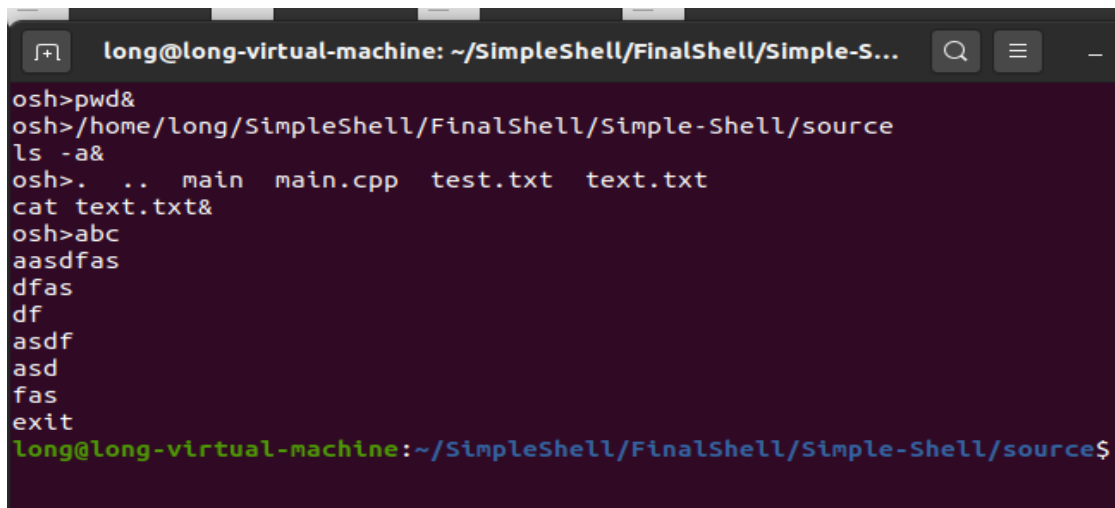
3.1. Simple commands with child processes

Hỗ trợ một số lệnh cơ bản trong Linux như `pwd`, `ls -a`, `cat`.

A terminal window titled 'long@long-virtual-machine: ~/SimpleShell/FinalShell/Simple-S...' with search, menu, and window control icons. The prompt is 'osh>'. The user enters 'pwd', and the output is '/home/long/SimpleShell/FinalShell/Simple-Shell/source'. Then 'osh>ls -a' is entered, showing a directory listing: '.', '..', 'main', 'main.cpp', 'test.txt', and 'text.txt'. Next, 'osh>cat text.txt' is entered, displaying the contents of 'text.txt': 'abc', 'aasdfas', 'dfas', 'df', 'asdf', 'asd', 'fas', and 'exit'. Finally, 'osh>exit' is entered, and the prompt changes to 'long@long-virtual-machine:~/SimpleShell/FinalShell/Simple-Shell/source\$ S'.

3.2. Simple command with '&'

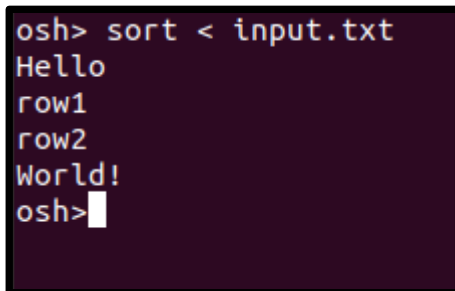
Dùng để chạy nền với một số lệnh cơ bản như `pwd`, `ls -a`, `cat`.

A terminal window with the same title and icons as the previous one. The prompt is 'osh>'. The user enters 'osh>pwd&', and the output '/home/long/SimpleShell/FinalShell/Simple-Shell/source' appears on the next line. Then 'osh>ls -a&' is entered, and the directory listing appears on the next line. Next, 'osh>. .. main main.cpp test.txt text.txt' is entered, and the output appears on the next line. Then 'osh>cat text.txt&' is entered, and the contents of 'text.txt' appear on the next line. Finally, 'osh>abc', 'osh>aasdfas', 'osh>dfas', 'osh>df', 'osh>asdf', 'osh>asd', 'osh>fas', and 'osh>exit' are entered sequentially, with their outputs appearing on the next lines. The final prompt is 'long@long-virtual-machine:~/SimpleShell/FinalShell/Simple-Shell/source\$ S'.

3.3. Input redirection

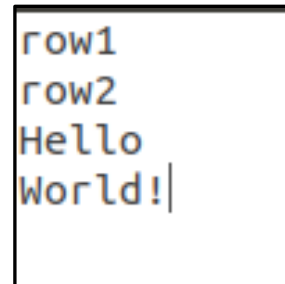
Sử dụng kí tự '<' để điều hướng Input.

Ví dụ: `osh > sort < input.txt`



```
osh> sort < input.txt
Hello
row1
row2
World!
osh>
```

Hình 3.3.1. Lệnh từ terminal



```
row1
row2
Hello
World!
```

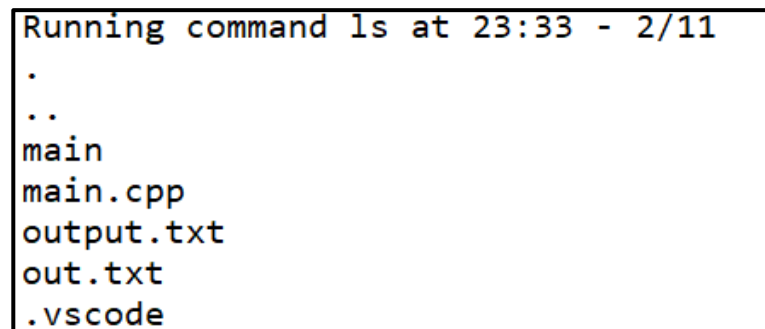
Hình 3.3.2. File input.txt

Các xâu có trong file "input.txt" đã được sắp xếp lại và in ra màn hình.

3.4. Output redirection

Sử dụng kí tự ">" để điều hướng Output.

Ví dụ: `ls -a > output.txt`



```
Running command ls at 23:33 - 2/11
.
..
main
main.cpp
output.txt
out.txt
.vscode
```

Hình 3.4.a: Dữ liệu trong file output.txt sau khi thực hiện lệnh.

3.5. Two commands with a pipe

Sử dụng kí tự '|' giữa hai lệnh.

Ví dụ: `osh> ls | sort`

```
osh>ls | sort
input.txt
main.cpp
output.txt
program
osh>
```

Output của lệnh `ls` được sử dụng làm input cho lệnh `sort` thông qua pipe.

3.6. Providing a history feature

Sử dụng "!!" để sử dụng lệnh gần nhất vừa được thực hiện.

Ví dụ:

`osh>ls`

`osh>!!`

```
osh>ls
main main.cpp output.txt out.txt
osh>!!
osh>ls
main main.cpp output.txt out.txt
osh>
```

4. Tài liệu tham khảo

Pipe system call: <https://www.geeksforgeeks.org/pipe-system-call/>

Dup2() system call: <https://www.geeksforgeeks.org/dup-dup2-linux-system-call/>

Các lệnh system call: <https://man7.org/linux/man-pages/>, <https://linux.die.net/>

Making linux shell c: <https://www.geeksforgeeks.org/making-linux-shell-c/>