Bài tập lớn – UNIX Shell

Yêu cầu: Viết một chương trình C thực hiện công việc giống như một giao diện Shell, ở đó chương trình nhận các lệnh từ người dùng và thực thi các lệnh này trên các tiến trình riêng biệt. Chương trình cài đặt hỗ trợ giao tiếp như pipe trong các hệ thống IPC đã học.

Để hoàn thành bài tập này, các em cần sử dụng các gọi hệ thống `fork()`, các biến thể của `exec()`, `wait()`, `dup2()` và `pipe()`. Chương trình có thể hoàn thành trên các hệ thống Unix, Linux, MacOS.

Giới thiệu chung

Một giao diện shell khi chạy sẽ đưa ra cho người dùng 1 lời nhắc nhập vào 1 lệnh. Ví dụ: `tuanlm>cat baitaplon.c` sẽ hiển thị nội dung của file `baitaplon.c` ra màn hình.

Một kỹ thuật để cài đặt giao diện shell là: tiến trình cha nhận lệnh từ phía người dùng (vd: `cat baitaplon.c`) và sau đó tạo tiến trình con để thực thi lệnh mà người dùng đã nhập vào từ bàn phím. Tiến trình cha sẽ đợi tiến trình con kết thúc trước khi thực thi lệnh tiếp theo.

Như đã biết, các Unix shell đều hỗ trợ các tiến trình chạy nền (background) hoặc chạy đồng thời. Để tiến trình cha và tiến trình con cùng chạy đồng thời, ta sẽ đặt dấu & vào cuối câu lệnh.

Tiến trình con sẽ thực thi câu lệnh nhận được từ tiến trình cha qua gọi hệ thống `exec()`.

Chương trình C giả lập Unix shell có thể được cài đặt như sau:

```
#include <stdio.h>
#include <unistd.h>
#define MAX LINE 80 /* Độ dài tối đa của lệnh */
int main(void)
{
      char *args[MAX LINE/2 + 1]; /* Tham số dòng lệnh */
      int run = 1; /* cờ xác định khi nào sẽ dừng chương trình*/
      while (run) {
           printf("tuanlm>");
           fflush(stdout);
           /**
           * Các bước tiếp theo sau khi nhận lệnh từ người dùng là:
           * (1) tạo tiến trình con dùng fork()
           * (2) Tiến trình con gọi execvp() để thực thi lệnh
           * (3) Tiến trình cha gọi wait() trừ khi lệnh chứa &
           */
      }
      return 0;
}
```

Chương trình hoàn chỉnh cần phải có:

a. *Tạo và thực thi lệnh trong tiến trình con*
b. *Ghi lại lịch sử lệnh*
c. *Cho phép tiến trình cha và con giao tiếp qua pipe.*

Hướng dẫn:

1. Tạo và thực thi lệnh trong tiến trình con
   Các lệnh người dùng đưa vào từ dòng lệnh cần phải lưu trong 1 mảng. Ví dụ: người dùng nhập câu lệnh `tuanlm>cat baitaplon.c` thì mảng `argv` sẽ như sau:

   ```
   argv[0] = "cat"
   argv[1] = "baitaplon.c"
   argv[2] = NULL
   ```

   mảng argv này sẽ làm tham số của hàm execvp với cú pháp như sau:

   ```
   execvp(char * command, char * param[]);
   ```

   trong đó, command là lệnh cần thực hiện và param là tham số của lệnh. Cần chắc chắn rằng người dùng có thêm dấu `&` vào cuối lệnh hay không để xác định tiến trình cha `wait` tiến trình con kết thúc.
2. Ghi lại lịch sử lệnh

   Chương trình cho phép thực thi lại các lệnh mà người dùng đã nhập bằng cách nhập vào dấu nhắc của hệ thống hai ký tự ! (!!). Ví dụ: nếu người dùng nhập câu lệnh ls -l, người đó có thể thực hiện lại câu lệnh này bằng cách nhập !!. Kết quả câu lệnh phải được hiển thị trên màn hình và câu lệnh được đặt vào hàng đợi lệnh (tạo lịch sử). Chương trình cần kiểm soát lỗi nếu không có lệnh nào trong hàng đợi.

3. Cho phép tiến trình cha và con giao tiếp qua pipe
   Ví dụ: `tuanlm>ls -l | more`
   Trong ví dụ trên, kết quả của câu lệnh ls -l là đầu vào của lệnh more. Hai lệnh ls và more sẽ phải chạy trên 2 tiến trình riêng biệt và giao tiếp với nhau qua pipe.
   Cách dễ nhất là tiến trình cha tạo tiến trình con để thực thi lệnh ls -l, tiến trình con này tiếp tục tạo tiến trình con của nó để thực thi lệnh less. Hai tiến trình này giao tiếp thông qua pipe. Gọi hệ thống dup2() được sử dụng để có thể cài đặt pipe trong tình huống này.

   Gọi hệ thống `dup2()` nhân bản một file descriptor tới một file descriptor khác. Ví dụ, nếu `fd` là một file descriptor tới file `out.txt`, lời gọi:

   ```
   dup2(fd, STDOUT_FILENO);
   ```

   nhân bản `fd` tới luồng ra chuẩn (màn hình). Điều này có nghĩa là, mọi thông tin được gửi tới luồng ra chuẩn cũng sẽ được gửi tới file `out.txt`.