Bài thực hành số 2

Lập trình multi-process

Ghi chú:

- Sinh viên nộp bài tại trang web: `www.cse.hcmut.edu.vn/portal`
- File nộp bài đặt tên là: *`lab2.tar.bz2`*
- Hạn chót nộp bài: *`4:00am 5/4/2010`*
- SV có thể chỉnh sửa source code mẫu nếu thấy cần thiết
- Mọi gian lận sẽ nhận điểm KHÔNG nếu bị phát hiện

1 Giới thiệu

1.1 Mục tiêu

Viết một chương trình C trên Linux để chạy các lệnh trên Linux sử dụng kĩ thuật lập trình multi-process.

1.2 Kiến thức cần biết

- Lập trình multiprocess
- Lập trình C trên Linux
- Makefile

1.3 Mô tả chương trình

Đầu vào (Input) của chương trình:

- Các lệnh cần chạy

Đầu ra (Output) của chương trình:

- Kết quả chạy được của các lệnh đó.

Chương trình lần lượt đọc các lệnh trong file input và chạy các lệnh đó. Khi gặp một lệnh mới, chương trình tạo thêm một process. Khi đó process cha ghi vào file output thông tin về số thứ tự của dòng lệnh, process con thực hiện lệnh và ghi kết quả của dòng lệnh đó vào file output. Thông tin của process cha luôn nằm trước thông tin process con.

1.4 Ý tưởng hiện thực chương trình

- Đọc các lệnh trong file input.
- Với mỗi lệnh đọc được, tạo một process con.
- Process cha ghi thông tin về số thứ tự dòng lệnh vào file output, sau đó chờ process con kết thúc.
- Process con thực hiện dòng lệnh, sau đó ghi kết quả của dòng lệnh này vào file output
- Cần hiện thực chương trình sao cho file output được ghi theo đúng thứ tự và định dạng.

1.5 Một số kỹ thuật lập trình

1.5.1 Xử lý thông số nhập vào từ chương trình

Một chương trình tốt thường cho phép người dùng thiết lập một vài thông số khi chạy chương trình, chẳng hạn khi thực hiện lệnh:

```
$ ls -R
```

người dùng đã truyền vào thông số `-R` để liệt kê các file và thư mục không chỉ trong thư mục hiện hành mà còn cả những thư mục con của thư mục hiện hành nếu có.

Sau đây là đoạn chương trình mẫu, sử dụng hàm `getopt()` để xử lý thông số -R ở trên:

```
int opt;
extern char *optarg;

while ((opt = getopt(argc, argv, "R")) != EOF) {
    switch (opt) {
        case 'R':
            // Option -R occurs
            // Proccess that option here
            break;
        default:
            // Other options
            break;
    }
}
```

1.5.2 Xử lý file cấu hình

File cấu hình thường ở dạng text và có cấu trúc (đơn giản). Ta thường sử dụng các hàm sau để xử lý những dạng file này:

- fopen: mở một file
- fclose: đóng file đã mở
- fscanf:: đọc/lọc thông tin trong file với định dạng cụ thể
- fgets: đọc một dòng trong file
- fprintf: ghi thông tin lên file

2 Yêu cầu

Chương trình sau khi biên dịch có tên là `runcommand`, hỗ trợ các thông số sau:

- `-h`: Hiển thị thông tin hướng dẫn sử dụng chương trình
- `-i` *filename*: Chọn file input. Đây là file chứa thông tin các lệnh cần chạy. Tên file input mặc định là *`input.txt`*
- `-o` *filename*: Chọn file output. Đây là file chứa kết quả của các lệnh. Tên file output mặc định là *`output.txt`*

Cú pháp chạy chương trình `runcommands`:

`runcommand [-h] [-i` *filename*`] [-o` *filename*`]`

Một số lưu ý khi xử lý option:

- Nếu dòng lệnh có option không nằm trong những option ở trên thì báo lỗi, thoát chương trình ngay lập tức và không quan tâm đến các option khác.
- Độ ưu tiên của option -h là cao nhất, sau đó là các option để chạy chương trình. Điều này có nghĩa là nếu dòng lệnh có option -h và các option chạy chương trình thì ta chỉ quan tâm đến option -h, và xem như không có các option còn lại.

3 Định dạng file input, output

3.1 File input

Mỗi dòng trong file input là một lệnh cần chạy. Ví dụ:

```
ls $HOME
cd /etc
pwd
```

Với file input mẫu ở trên, chương trình xem như file input có 3 lệnh cần được thực thi. Lệnh `ls $HOME`, lệnh `cd /etc` và lệnh `pwd`.

3.2 File output

- Kết quả của các lệnh trong file input sẽ được ghi vào file output.
- Nội dung của chúng phải được ghi theo đúng thứ tự của các lệnh trong file input.
- Trước mỗi kết quả có dòng chú thích ghi đây là kết quả của dòng lệnh nào trong file input (số thứ tự bắt đầu là 0)
- Giữa 2 kết quả của 2 dòng lệnh cách nhau bằng một dòng trống (không có khoảng trắng, dấu tab, ... ở dòng này)
- Nếu một lệnh không xuất quả, vẫn phải ghi dòng chú thích và cách với các kết quả khác bởi một dòng trống.
- Nếu lệnh phía trước làm thay đổi cấu trúc cây thư mục (mkdir, rm, rmdir, touch, ...) thì kết quả của nó sẽ ảnh hưởng đến lệnh phía sau, nếu không (ls, pwd, cd, ...) thì kết quả của nó không ảnh hưởng đến dòng lệnh phía sau.

```
#command 0                      // Dòng chú thích do process cha ghi vào
bin
Desktop
Documents

                                // Dòng trống
#command 1                      // Lệnh 1 không tạo output

#command 2
/home/ntnguyen/lab2/lab2        // Kết quả do process con ghi vào
```