|  |
| --- |
| Họ và tên: Nguyễn Nguyên Khang  Mã số sinh viên: 22520623 |
| Họ và tên: Phùng Vũ Nguyên Khang  Mã số sinh viên: 22520626 |
| Họ và tên: Huỳnh Tống Đăng Khoa  Mã số sinh viên: 22520670 |
| Lớp: IT007.O14.1 |

HỆ ĐIỀU HÀNH  
BÁO CÁO LAB 6

**CHECKLIST**

**6.4. BÀI TẬP THỰC HÀNH**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Câu 1** | **Câu 2** | **Câu 3** | **Câu 4** | **Câu 5** |
| **Trình bày giải thuật** |  |  |  |  |  |
| **Chụp hình minh chứng (chạy ít nhất 3 lệnh)** |  |  |  |  |  |
| **Giải thích code, kết quả** |  |  |  |  |  |

**Tự chấm điểm:** 10

*\*Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:* ***<Tên nhóm>\_LAB6.pdf***

**6.4. BÀI TẬP THỰC HÀNH**

# Câu 1

**Trình bày giải thuật:**

* Đầu tiên để chạy một lệnh trong một tiến trình con riêng biệt, ta dùng hàm **fork()** để tạo ra một tiến trình con.
* Dùng một lệnh if để kiểm tra xem fork() thành công hay không, nếu thành công thì kiểm tra nếu (pid == 0) thì ta xử lí các lệnh chạy trong tiến trình con
* Tạo ra một mảng động để lưu lại câu lệnh của người dùng nhập vào (ở đây là mảng arguments)
* Tạo thêm một con trỏ token để lưu lại các phần được tách ra của các câu lệnh do người dùng nhập vào bằng lệnh strtok
* Tạo biến i = 0 để đánh dấu index của mảng arguments.
* Dùng một vòng lặp while và lệnh strtok tách câu lệnh đó thành nhiều thành phần khác nhau, rồi lưu vào mảng arguments (ví dụ lệnh cat abc.txt sẽ tách thành “cat” và “abc.txt”)
* Kết thúc chuỗi lệnh bằng cách gán phần tử thứ i của mảng arguments = NULL
* Dùng hàm execvp với tham số là phần lệnh đã tách ở trên, ví dụ như sau khi đã tách lệnh cat abc.txt thành “cat” và “abc.txt”, ta sẽ truyền tham số vào hàm execvp như sau execvp(“cat”, “abc.txt”)
* Nếu pid khác 0 ta thực hiện lệnh waitpid để tiến trình cha đợi tiến trình con thực hiện xong.

**Chụp hình minh chứng:**

* **Hiện thực code:**

**A screen shot of a computer program

Description automatically generated**

**A computer screen with text on it

Description automatically generated**

* **Kết quả:**

**A computer screen shot of a computer

Description automatically generated**

* **Giải thích kết quả:**
  + **Ở kết quả thứ nhất (lệnh ls),** lệnh strtok sẽ tách câu lệnh sao cho mảng arguments[“ls”, NULL], sử dụng lệnh execvp ta sẽ thực hiện lệnh ls (in ra danh sách các tệp và thư mục trong thư mục hiện tại mà không cung cấp thông tin chi tiết về chúng)
  + **Ở kết quả thứ hai (lệnh ls -l),** lệnh strtok sẽ tách câu lệnh sao cho mảng arguments[“ls”, ”-l”, NULL], sử dụng lệnh execvp ta sẽ thực hiện lệnh ls -l (cung cấp một bảng thông tin chi tiết về các tệp và thư mục, bao gồm quyền truy cập, số lượng liên kết, người sở hữu, nhóm sở hữu, kích thước, và thời gian sửa đổi)
  + **Ở kết quả thứ ba (lệnh cat abc.txt),** lệnh strtok sẽ tách câu lệnh sao cho mảng arguments[“cat”, ”abc.txt”, NULL], sử dụng lệnh execvp ta sẽ thực hiện lệnh cat abc.txt (in ra nội dung trong file abc.txt)

# Câu 2

# Câu 3

**Trình bày giải thuật:**

* Ta thực hiện tương tự như đoạn code ở bài 2, tuy nhiên sau khi đã tách các lệnh và gán NULL vào vị trí cuối cùng của mảng arguments, ta bắt đầu kiểm tra xem có toán tử chuyển hướng đầu ra trong mảng arguments không rồi sử dụng dup2 để sao chép một file descriptor thành một file descriptor khác, cách làm như sau:
  + **Kiểm tra Chuyển hướng Đầu ra ('>') và Đầu vào ('<'):** Hai biến, redirectOutput và redirectInput, được sử dụng để lưu vị trí của toán tử chuyển hướng đầu ra và đầu vào tương ứng. Dùng vòng lặp for kiểm tra qua các đối số của lệnh và ghi lại vị trí bằng hai biến redirectInput và redirectOutput nếu phát hiện trong mảng arguments có tồn tại '>' hoặc '<'.
  + **Chuyển hướng Đầu ra ('>'):** Nếu redirectOutput > 0, có nghĩa là đã tìm thấy toán tử '>'. Mở một tệp để ghi hoặc tạo một tệp mới nếu nó chưa tồn tại bằng cách sử dụng open. Chuyển hướng đầu ra tiêu chuẩn (STDOUT\_FILENO) đến tệp đã mở bằng dup2. Đóng tệp để giải phóng tài nguyên hệ thống. Loại bỏ '>' và tên tệp từ danh sách đối số bằng một vòng lặp for.
  + **Chuyển hướng Đầu vào ('<'):** Nếu redirectInput lớn hơn 0, có nghĩa là đã tìm thấy toán tử '<'. Mở một tệp để đọc bằng open. Chuyển hướng đầu vào tiêu chuẩn (STDIN\_FILENO) đến tệp đã mở bằng dup2. Đóng tệp để giải phóng tài nguyên hệ thống. Loại bỏ '<' và tên tệp từ danh sách đối số bằng một vòng lặp for.
  + Sau đó, ta thực hiện bình thường câu lệnh qua hàm execvp như bài 1, lúc này đầu vào (hoặc đầu ra) sẽ được chuyển tiếp như ta mong muốn nhờ hàm dup2.

**Chụp hình minh chứng:**

* **A screen shot of a computer program

  Description automatically generatedHiện thực code:**

**A screen shot of a computer program

Description automatically generated**

* **Kết quả:**

A screenshot of a computer

Description automatically generated

**Giải thích kết quả: Ta dùng lệnh cat để kiểm tra nội dung file sau khi chuyển hướng**

* **Ở kết quả thứ nhất (lệnh ls -l > output.txt),** lệnh strtok sẽ tách câu lệnh sao cho mảng arguments [“ls”, “-l”, NULL], sử dụng lệnh execvp ta sẽ thực hiện lệnh ls -l (in ra danh sách các tệp và thư mục trong thư mục hiện tại mà không cung cấp thông tin chi tiết về chúng) sau đó chuyển hướng đầu ra vào tệp output.txt qua lệnh dup2(fd, STDOUT\_FILENO) (fd là file output.txt).
* **Ở kết quả thứ nhất (cat < abc.txt),** lệnh strtok sẽ tách câu lệnh sao cho mảng arguments [“cat”, NULL], sử dụng lệnh execvp ta sẽ thực hiện lệnh cat để đọc nội dung từ tệp abc.txt qua toán tử chuyển hướng đầu vào và lệnh dup2(fd, STDIN\_FILENO) (fd là file abc.txt).
* **Ở kết quả thứ nhất (lệnh sort < abc.txt > output\_sorted.txt),** lệnh strtok sẽ tách câu lệnh sao cho mảng arguments [“sort”, NULL], sử dụng lệnh execvp ta sẽ thực hiện lệnh sort để sắp xếp nội dung từ tệp abc.txt qua toán tử chuyển hướng đầu vào và lệnh dup2(fd, STDIN\_FILENO) (fd là file abc.txt) sau đó chuyển hướng đầu ra vào tệp output\_sorted.txt qua lệnh dup2(fd, STDOUT\_FILENO) (fd là file > output\_sorted.txt).

**Câu 4**

**Trình bày cách làm**:

* **Hàm execute pipeline** nhận 2 tham số đầu vào là một command\_list chứa các lệnh cần thực thi trong 1 command lớn hơn có chứa các pipes và 1 biến số nguyên num\_commands chứa số lượng câu lệnh.
* Đầu tiên chúng ta tiến hành khai báo mảng hai chiều pipes[num\_commands-1][2] lưu thông tin các ống dẫn do 1 lệnh lớn có num\_commands lệnh nhỏ hơn sẽ có num\_commands-1 ống dẫn, mỗi phần tử trong mảng có 2 phần tử con chứa các filedescriptor để đọc nội dung từ lệnh trước đó và ghi nội dung vào ống dẫn cho lệnh tiếp theo.
* Bên trong vòng lặp for, chúng ta dùng câu lệnh if để kiểm tra điều kiện i < num\_commands-1 (i có phải là phần tử cuối cùng không), nếu i không phải là phần tử cuối cùng thì phải có 1 ống dẫn nối lệnh thứ i đến lệnh kế tiếp.
* Chúng ta tiếp tục kiểm tra điều kiện pipe(pipes[i])== -1, pipe() là hàm dùng để tạo ra 1 cặp filedescriptor để ống dẫn pipes[i] có thể đọc và ghi từ các lệnh, hàm này sẽ trả về -1 nếu tạo ống dẫn thất bại. Nếu việc tạo ống dẫn thất bại, hàm perror() được gọi để hiển thị thông báo lỗi ra màn hình và hàm exit() được gọi để kết thúc chương trình với trạng thái lỗi.
* Nếu việc tạo ống dẫn không bị lỗi, ta tiếp tục thực thi chương trình, biến pid có kiểu dữ liệu pid\_t được khởi tạo và hàm fork() được gọi để tạo ra 1 tiến trình con thực thi câu lệnh đang xét.
* Nếu (pid==0) tức là ta đang trong tiến trình con, ta tiếp tục kiểm ra điều kiện i>0 (lệnh hiện tại có phải lệnh đầu tiên hay không). Nếu không phải là lệnh đầu tiên thì đầu ra của lệnh trước đó phải được ghi vào ống dẫn hướng vào đầu vào của lệnh hiện tại. Do đó, hàm dup2(pipes[i-1][0], STDIN\_FILENO) được gọi để nhân bản filedescriptor ứng với đầu đọc của ống dẫn nối đến lệnh trước đó vào filedescriptor input tiêu chuẩn STDIN\_FILENO. Điều này sẽ thiết lập 1 input tiêu chuẩn cho lệnh hiện tại kết nối với output của lệnh trước đó thông qua ống dẫn.
* 2 dòng lệnh close() tiếp theo dùng để đóng cả filedescriptor đọc và ghi của ống dẫn nối với lệnh trước đó vì lúc này lệnh hiện tại sẽ đọc từ filedescriptor đã được nhân bản trước đó.
* Ta tiếp tục kiểm tra điều kiện i < num\_commands-1 (lệnh hiện tại có phải là lệnh cuối cùng không). Nếu không phải là lệnh cuối cùng thì đầu ra của lệnh hiện tại phải được kết nối với đầu vào của lệnh tiếp theo thông qua ống dẫn.
* Ta tiếp tục gọi hàm dup2(pipes[i][1], STDOUT\_FILENO) để nhân bản filedescriptor ứng với đầu ghi của ống dẫn nối giữa lệnh hiện tại và lệnh tiếp theo vào filedescriptor output tiêu chuẩn STDOUT\_FILENO. Điều này thiết lập output tiêu chuẩn của lệnh hiện tại sẽ nối vào input của lệnh kế tiếp thông qua ống dẫn.
* 2 dòng lệnh close() tiếp theo dùng để đóng cả filedescriptor đọc và ghi của ống dẫn nối với lệnh hiện tại, vì lúc này lệnh hiện tại sẽ ghi vào filedescriptor đã được nhân bản trước đó.
* Tiếp theo chúng ta gọi hàm execvp để thực thi câu lệnh hiện tại đã được lưu trong command\_list[i], nếu execvp thực thi không thành công, hàm perror sẽ được gọi để xuất thông báo lỗi và hàm exit() để kết thúc chương trình.
* Trong điều kiện pid<0 (hàm fork() thất bại), hàm perror() sẽ được gọi để xuất thông báo lỗi và hàm exit() để kết thúc chương trình trong trạng thái lỗi.
* Sau khi tiến trình con được tạo và lệnh hiện tại đã được thực thi, nếu lệnh hiện tại không phải là lệnh đầu tiên (i > 0), cả filedescriptor đọc và ghi của ống dẫn nối lệnh trước đó với lệnh hiện tại được đóng lại bằng 2 hàm close().
* Dòng wait(NULL) trong tiến trình cha để chờ tiến trình con thực thi xong.
* **Bên trong hàm executeCommand** nhận đối số đầu vào là command cần thực hiện, ta tiến hành khai báo thêm biến số nguyên num\_commands được khởi tạo giá trị ban đầu = 0 (số lệnh nhỏ trong lệnh có ống dẫn được người dùng nhập vào) và một con trỏ của con trỏ command\_list dùng để lưu các lệnh con này.
* Trong vòng lặp while đang xử lý các thành phần từ câu lệnh được người dùng nhập vào, nếu phát hiện có ký tự “|” (ống dẫn) bằng việc kiểm tra điều kiện if (strcmp(token, “|”)==0), ta tiến hành các xử lý cho câu lệnh có chứa ống dẫn.
* Gán giá trị arguments[i]= NULL để kết thúc danh sách các tham số của lệnh hiện tại.
* Dùng hàm malloc để cấp phát bộ nhớ một giá trị i+1 (do giá trị ban đầu của i là 0 nên i+1 là số lượng thực sự các thành phần của lệnh )lần kích thước của kiểu dữ liệu char\* cho command\_list[num\_commands] (num\_commands chạy từ 0).
* Gọi hàm memcpy() đểu lưu danh sách tham số (arguments) vào command\_list[num\_commands]
* Gán giá trị NULL cho command\_list[num\_commands][i] để kết thúc danh sách tham số của command\_list[num\_commands] (các tham số đã được lưu từ command\_list[num\_commands][0] đến command\_list[num\_commands][i-1] tương tự như từ arguments[0] đến arguments[i-1] ở thời điểm hiện tại).
* Ta tiến hành cộng thêm 1 cho giá trị num\_commands và gán lại i=0.
* Có thể thấy, cứ mỗi lần tìm thấy một ký tự “|” (ống dẫn), ta xem như đã đọc hết 1 lệnh con và lưu lệnh này vào command\_list, gán lại i=0 để ta tiếp tục đọc lệnh con tiếp theo trong dòng lệnh lớn.
* Ngược lại, nếu trong quá trình duyệt qua vòng lặp không tìm thấy bất cứ ký tự “|” (ống dẫn) nào, ta lần lượt lưu các tham số của lệnh vào arguments và cập nhật biến i như bình thường.
* Sau khi đã đọc hết dòng lệnh do người dùng nhập vào, ta cập nhật arguments[i] = NULL xem như kết thúc danh sách tham số của dòng lệnh.
* **Với phần thực thi câu lệnh**, ta kiểm tra điều kiện i>0 (lệnh hiện tại có tham số đầu vào), lúc này có 2 trường hợp, nếu là câu lệnh thông thường không sử dụng ống dẫn thì sẽ không tìm thấy ống dẫn, nếu là câu lệnh có sử dụng ống dẫn thì ở lệnh con cuối cùng sẽ không có ống dẫn ở đuôi (nếu tìm thấy ống dẫn ở cuối lệnh sẽ gán lại biến i = 0) nên cả 2 trường hợp này đều dẫn đến i > 0, dù ở trường hợp nào ta cũng sẽ lưu lệnh này vào command\_list (việc có thực thi trên ống dẫn đối với lệnh này hay không sẽ được quyết định ở bên dưới).
* Thêm lệnh hiện tại vào command\_list:
* Ta gán arguments[i]=NULL để kết thúc danh sách tham số lưu trong arguments.
* Ta tiến hành cấp phát động cho command\_list[num\_commands] một vùng nhớ có kích thước i+1 (số lượng tham số thực sự) lần kích thước của kiểu dữ liệu char\*.
* Sau đó ta copy danh sách tham số từ arguments vào command\_list[num\_commands] bằng hàm memcpy().
* Ta gán giá trị command\_list[num\_commands][i]=NULL để đánh dấu kết thúc danh sách tham số của lệnh con trong command\_list.
* Ta tiến hành cập nhật giá trị num\_commands thêm 1.
* Lúc này ta có nếu câu lệnh không sử dụng ống dẫn thì nó cũng được thêm vào command\_list và num\_commands nhận giá trị là 1. Nếu câu lệnh có sử dụng ống dẫn thì tất cả các lệnh con đã được thêm vào command\_list và num\_command sẽ nhận giá trị là n (ứng với trường hợp có n-1 ống dẫn) lớn hơn 1.
* Do đó ta kiểm tra điều kiện num\_commands > 1, nếu đúng, ta gọi hàm execute\_pipeline đã được định nghĩa ở trên với đối số truyền vào là command\_list và num\_commands để thực thi câu lệnh.
* Nếu điều kiện sai, câu lệnh sẽ được thực thi như một lệnh thông thường.

**Chụp hình minh chứng**:

* **Hiện thực code:**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự độngẢnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

* **Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, đen và trắng

  Mô tả được tạo tự độngẢnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, thông tin

  Mô tả được tạo tự độngẢnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

  Mô tả được tạo tự độngKết quả chạy:**

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, số

Mô tả được tạo tự động

* **Giải thích kết quả**:
  + **Ở kết quả thứ nhất**, khi chạy lệnh ps aux | grep home, lệnh con ps aux sẽ hiển thị ra thông tin về các tiến trình đang chạy trong hệ thống, thông tin này sẽ được pipe nối vào đầu vào của lệnh kế tiếp là grep home, lệnh này sẽ tìm kiếm vị trí các từ home từ output của lệnh ps aux và hiện thị ra màn hình.
  + **Ở kết quả thứ hai,** khi chạy lệnh ps aux | grep vscode, tương tự, pipe sẽ nối đầu ra của ps aux vào đầu vào của lệnh con grep vscode, lệnh này sẽ tìm kiếm vị trí các từ vscode và hiển thị ra màn hình.
  + **Ở kết quả thứ ba,** khi chạy lệnh ps aux | grep home | grep token, pipe sẽ nối đầu ra của ps aux vào đầu vào của grep home và nối đầu ra của grep home vào đầu vào của grep token, trong số các vị trí tìm thấy từ home từ lệnh ps aux, chương trình tiếp tục tìm ra các vị trí có từ token và hiển thị ra màn hình.
  + **Ở kết quả thứ tư,** khi chạy lệnh ls -l | grep c | wc -l, pipe sẽ nối đầu ra của ls -l vào đầu vào của grep c và nối đầu ra của grep c vào đầu vào của wc -l, lệnh ls sẽ cho ra danh sách các thành phần có trong thư mục hiện thời và grep c sẽ xác định vị trí các thành phần có chứa ký tự “c”, wc -l sau đó sẽ đếm số dòng từ kết quả của grep c và in ra màn hình (ở đây kết quả cụ thể là 8 dòng).