|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **MSSV** | **Lớp** |
| Lại Quan Thiên | 22521385 | IT007.O211.1 |
| Đặng Đức Tài | 22521270 |
| Mai Nguyễn Nam Phương | 22521164 |
| Phùng Trần Thế Nam | 21522366 |

HỆ ĐIỀU HÀNH  
BÁO CÁO LAB 6

**CHECKLIST**

**6.4. BÀI TẬP THỰC HÀNH**

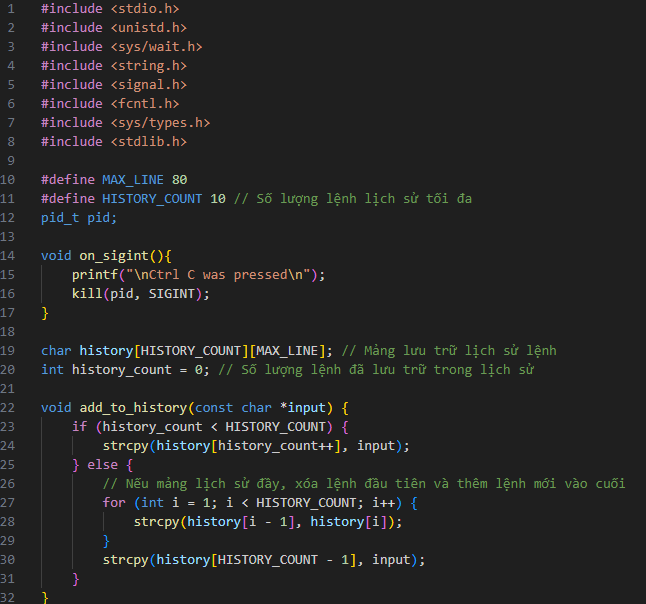
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Câu 1** | **Câu 2** | **Câu 3** | **Câu 4** | **Câu 5** |
| **Trình bày giải thuật** |  |  |  |  |  |
| **Chụp hình minh chứng (chạy ít nhất 3 lệnh)** |  |  |  |  |  |
| **Giải thích code, kết quả** |  |  |  |  |  |

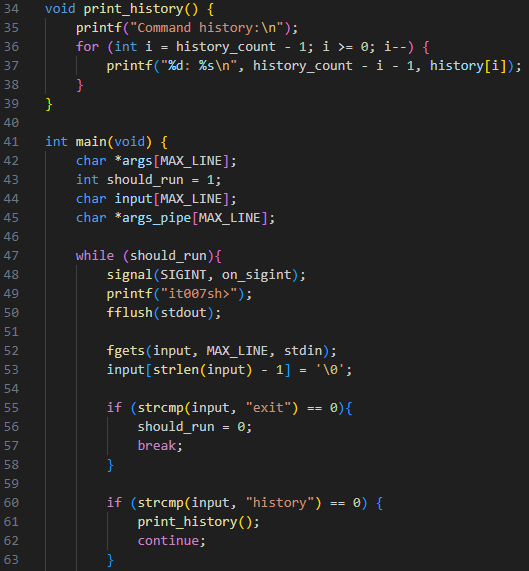
**Tự chấm điểm:** 10/10

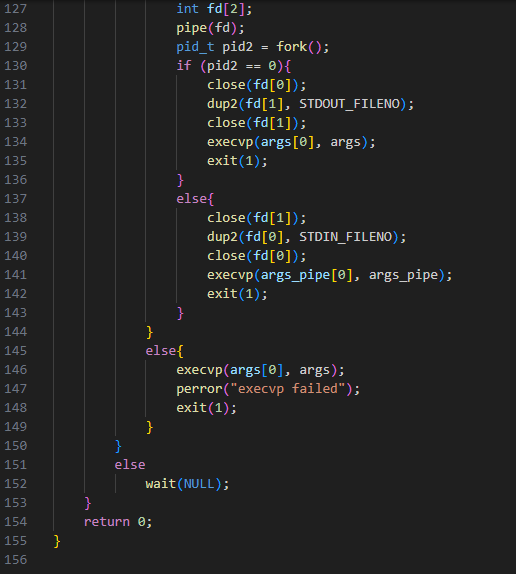
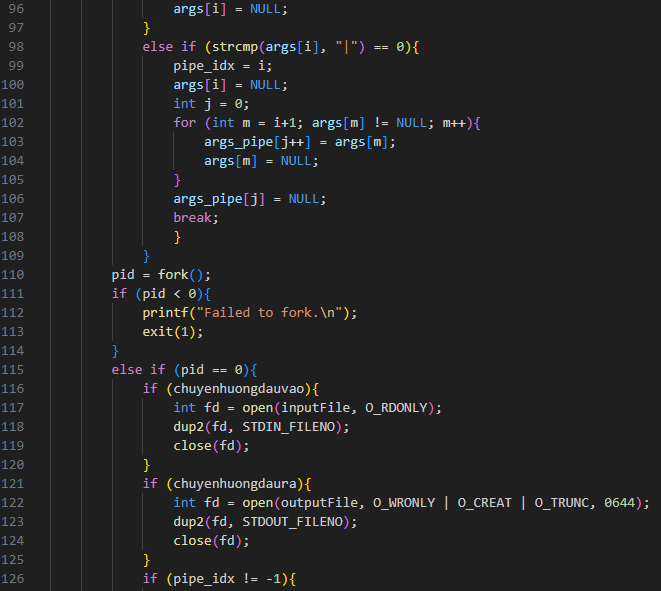
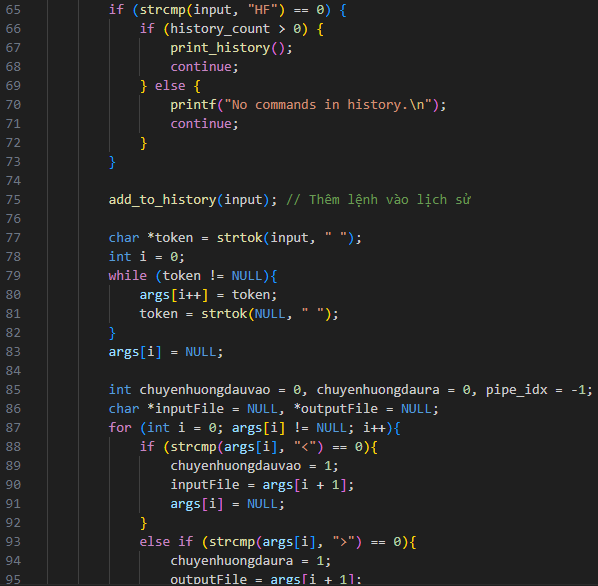
*\*Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:* ***<Tên nhóm>\_LAB6.pdf***

**6.4. BÀI TẬP THỰC HÀNH**

**\* FULL SOURCE CODE:**

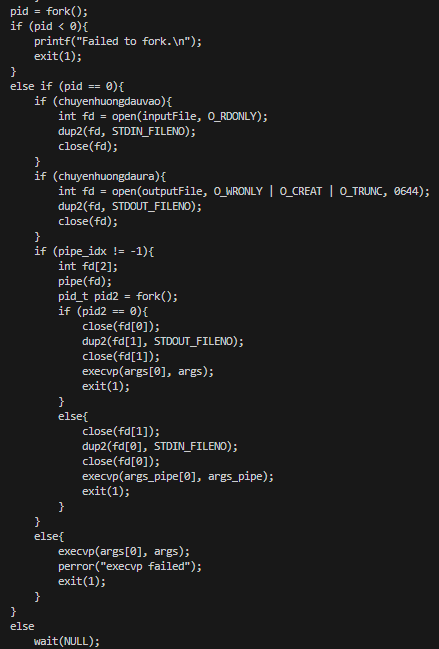






# Câu 1

**CODE:**



**GIẢI THÍCH CODE:**

**1. Khởi tạo tiến trình con bằng fork():** Hàm fork() tạo ra một tiến trình con từ tiến trình cha. Nếu fork() trả về giá trị âm, điều này có nghĩa là tiến trình cha không thể tạo được tiến trình con, và chương trình sẽ kết thúc với thông báo lỗi.

**2. Thực thi lệnh trong tiến trình con:**Nếu fork() trả về 0, điều này có nghĩa là mã đang chạy trong tiến trình con. Tiến trình con sẽ thực thi lệnh được nhập bởi người dùng.

**3. Chuyển hướng đầu vào:** Nếu lệnh yêu cầu chuyển hướng đầu vào (kí hiệu <), tiến trình con sẽ mở tệp chỉ định ở chế độ chỉ đọc và thay thế đầu vào chuẩn (stdin) bằng mô tả tệp của tệp này. Điều này có nghĩa là dữ liệu từ tệp sẽ được dùng làm đầu vào cho lệnh.

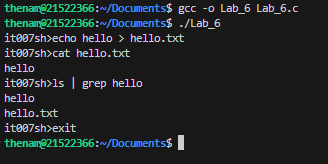
**4. Chuyển hướng đầu ra:** Nếu lệnh yêu cầu chuyển hướng đầu ra (kí hiệu >), tiến trình con sẽ mở hoặc tạo tệp chỉ định ở chế độ ghi, và thay thế đầu ra chuẩn (stdout) bằng mô tả tệp của tệp này. Điều này có nghĩa là dữ liệu đầu ra của lệnh sẽ được ghi vào tệp thay vì hiển thị trên màn hình.

**5. Sử dụng pipe (|):** Nếu lệnh chứa kí hiệu pipe (|), tiến trình con sẽ tạo một tiến trình con mới để thực thi lệnh trước kí hiệu pipe và chuyển hướng đầu ra của lệnh này đến đầu vào của lệnh sau pipe. Pipe cho phép kết nối đầu ra của một lệnh với đầu vào của lệnh khác.

**6. Thực thi lệnh:** Tiến trình con sẽ thực thi lệnh bằng cách sử dụng hàm execvp(), thay thế nội dung của tiến trình con bằng lệnh mới. Nếu execvp() thất bại, chương trình sẽ in ra thông báo lỗi và thoát.

**7. Tiến trình cha đợi tiến trình con hoàn thành:** Sau khi tạo tiến trình con, tiến trình cha sẽ đợi tiến trình con hoàn thành trước khi tiếp tục vòng lặp để đọc và thực thi lệnh tiếp theo. Điều này đảm bảo rằng shell sẽ không chấp nhận lệnh mới từ người dùng cho đến khi lệnh hiện tại hoàn thành.

**CHẠY CODE:**

  
**1.Tạo và ghi vào tệp hello.txt: echo hello > hello.txt**

-> Tạo tệp hello.txt và ghi chuỗi "hello" vào đó.

**2.Hiển thị nội dung của hello.txt: cat hello.txt**

-> Hiển thị nội dung của tệp hello.txt, kết quả là "hello".

**3.Liệt kê và tìm kiếm các tệp có tên chứa "hello": ls | grep hello**

-> Liệt kê các tệp và thư mục trong thư mục hiện tại, rồi tìm và hiển thị các tên có chứa "hello".

# Câu 2

**CODE:**

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

Mô tả được tạo tự động

**GIẢI THÍCH:**

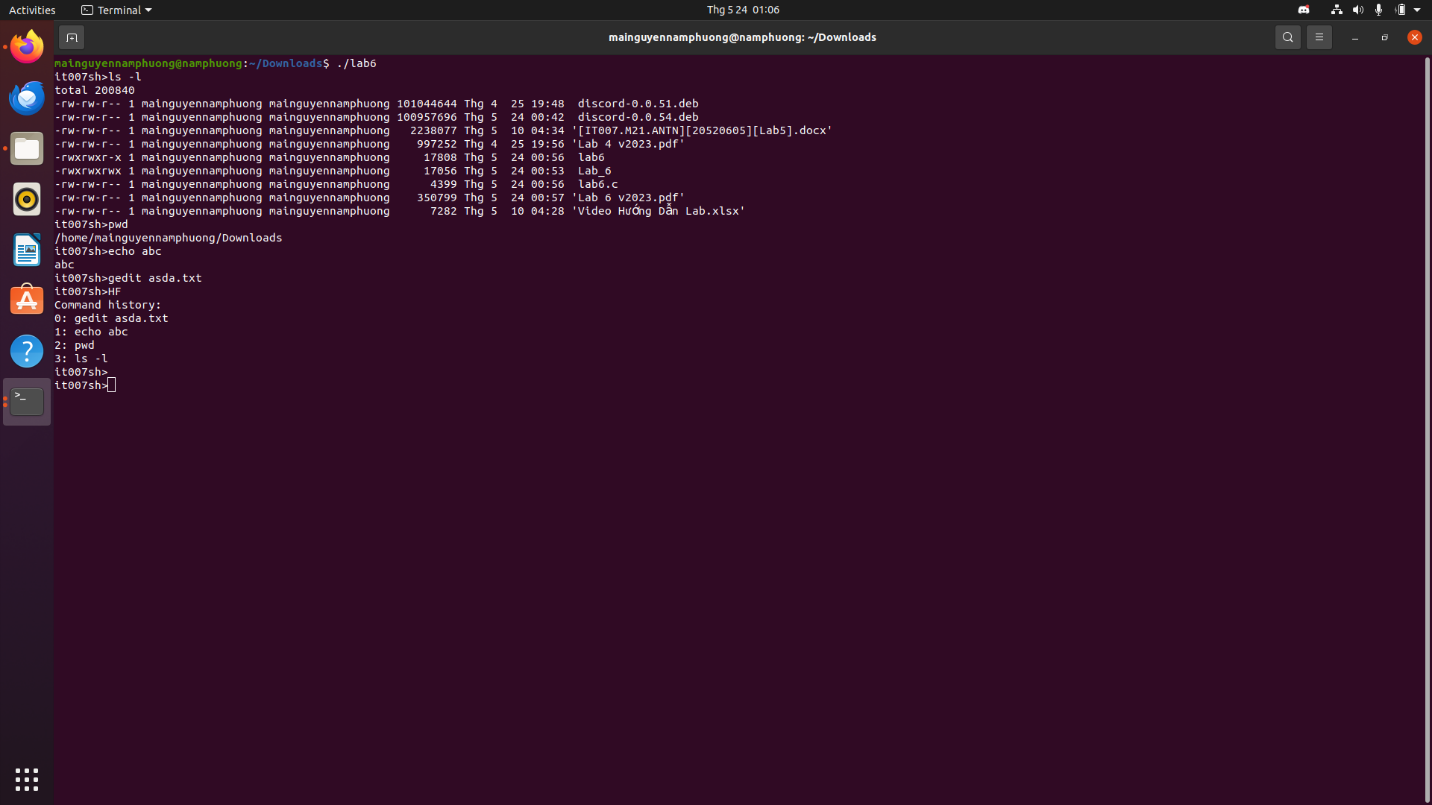
* Đầu tiên ta sẽ tạo mảng ‘history’ để lưu trữ các lệnh đã thực thi.
* Tạo 1 biến ‘history\_count’ để giữ số lượng lệnh trong lịch sử.
* Hàm ‘add\_to\_history’ dùng để ta push các lệnh đã nhập vào trong mảng lưu, nếu mảng đã đầy thì ta sẽ xoá lệnh đầu tiên và thêm lệnh mới vào
* Hàm ‘print\_history’ sẽ cho ta xem các lệnh đã được ta nhập ( lưu trong mảng history)

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

Mô tả được tạo tự động

* Đây là hàm xử lí trong hàm main, dễ dàng nhận thấy lệnh sau khi được nhập sẽ được lưu lại nhờ hàm add\_to\_history(), sau đó nếu input của người dùng là “HF” thì ta sẽ gọi kiểm tra biến history\_count, nếu nó > 0 thì chứng tỏ đã có lệnh được thực hiện, khi đó ta sẽ gọi hàm ‘print\_history’ nhằm cho người dùng xem lịch sử các câu lệnh

**KẾT QUẢ:**



1. **Câu 3**

**CODE:**

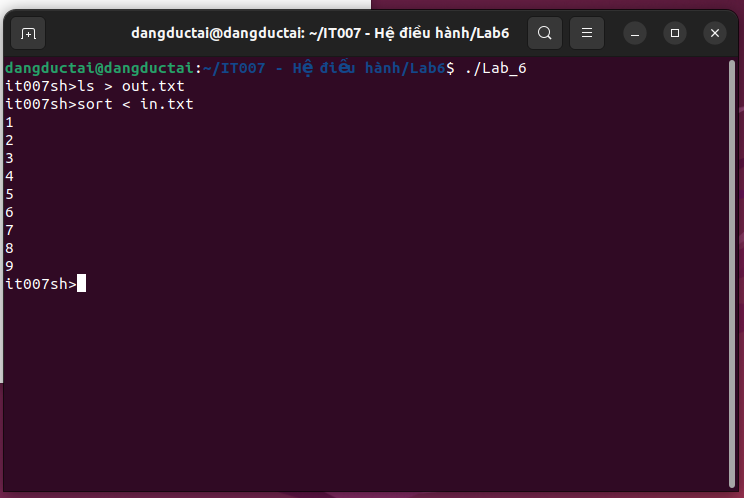
A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**KẾT QUẢ:**



**GIẢI THÍCH:**

- Kiểm tra và thiết lập các cờ chuyển hướng:

* *chuyenhuongdauvao* và *chuyenhuongdaura* là các cờ (flag) để xác định xem có yêu cầu chuyển hướng đầu vào hoặc đầu ra không.
* *inputFile* và *outputFile* lưu trữ tên tệp sẽ được sử dụng cho chuyển hướng đầu vào và đầu ra.
* Vòng lặp for duyệt qua các phần tử trong args để tìm các ký tự <, >, và |.

- Nếu < được tìm thấy:

* Đặt *chuyenhuongdauvao* thành 1 để đánh dấu rằng có chuyển hướng đầu vào.
* Lưu tên tệp đầu vào vào *inputFile*.
* Đặt args[i] thành NULL để ngắt chuỗi tại vị trí này.

- Nếu > được tìm thấy:

* Đặt *chuyenhuongdaura* thành 1 để đánh dấu rằng có chuyển hướng đầu ra.
* Lưu tên tệp đầu ra vào *outputFile*.
* Đặt args[i] thành NULL để ngắt chuỗi tại vị trí này.

- Nếu | được tìm thấy:

* Đặt pipe\_idx thành vị trí của |.
* Ngắt chuỗi args tại vị trí này bằng cách đặt args[i] thành NULL.
* Sao chép các lệnh sau | vào args\_pipe.

- Chuyển hướng đầu vào (Nếu *chuyenhuongdauvao* là 1):

* Mở tệp *inputFile* với quyền đọc *(O\_RDONLY).*
* Dùng *dup2(fd, STDIN\_FILENO)* để chuyển hướng đầu vào từ tệp này thay vì từ bàn phím *(STDIN\_FILENO).*
* Đóng tệp mô tả fd sau khi đã chuyển hướng.

- Chuyển hướng đầu ra (Nếu *chuyenhuongdaura* là 1):

* Mở tệp *outputFile* với quyền ghi *(O\_WRONLY)*. Nếu tệp không tồn tại, tạo tệp mới *(O\_CREAT)* và xóa nội dung tệp nếu nó đã tồn tại *(O\_TRUNC)*.
* Dùng *dup2(fd, STDOUT\_FILENO)* để chuyển hướng đầu ra đến tệp này thay vì màn hình *(STDOUT\_FILENO)*.
* Đóng tệp mô tả fd sau khi đã chuyển hướng.

1. **Câu 4**

**CODE & GIẢI THÍCH:**

**A computer screen shot of text

Description automatically generated**

Ảnh 1

**\* Ảnh 1:**

1. Khởi tạo biến pipe\_idx: Lưu vị trí của ký tự pipe (|) trong mảng lệnh args. Ban đầu gán giá trị -1 để kiểm tra sau này.
2. Duyệt qua mảng args để tìm các lệnh đặc biệt:

* Vòng lặp for duyệt qua từng phần tử của args (mảng chứa các lệnh và tham số đã tách từ đầu vào của người dùng).
* Pipe (|):
  + - Nếu phần tử hiện tại là |, lưu chỉ mục của pipe vào pipe\_idx, đặt args[i] thành NULL để ngắt lệnh tại vị trí đó.
    - Sau đó, sao chép các phần tử sau | vào mảng args\_pipe.

**A screen shot of a computer program

Description automatically generated**

Ảnh 2

**\* Ảnh 2:**

1. **Kiểm tra lệnh pipe**: Nếu **pipe\_idx** khác **-1**, điều này có nghĩa là người dùng đã nhập lệnh có chứa pipe (**|**).
2. **Tạo pipe**: Sử dụng **pipe(fd)** để tạo một pipe. **fd[0]** là đầu đọc, **fd[1]** là đầu ghi.
3. **Tạo tiến trình con thứ hai (pid2)**:
   * Gọi **fork()** để tạo tiến trình con thứ hai.
   * Nếu **pid2 == 0** (trong tiến trình con thứ hai):
     + Đóng đầu đọc của pipe (**fd[0]**).
     + Sử dụng **dup2(fd[1], STDOUT\_FILENO)** để chuyển đầu ra chuẩn sang đầu ghi của pipe.
     + Đóng đầu ghi của pipe (**fd[1]**).
     + Gọi **execvp(args[0], args)** để thực thi lệnh trước pipe.
     + Nếu lệnh **execvp** thất bại, gọi **exit(1)** để kết thúc tiến trình con.
4. **Trong tiến trình cha**:
   * Đóng đầu ghi của pipe (**fd[1]**).
   * Sử dụng **dup2(fd[0], STDIN\_FILENO)** để chuyển đầu vào chuẩn sang đầu đọc của pipe.
   * Đóng đầu đọc của pipe (**fd[0]**).
   * Gọi **execvp(args\_pipe[0], args\_pipe)** để thực thi lệnh sau pipe.
   * Nếu lệnh **execvp** thất bại, gọi **exit(1)** để kết thúc tiến trình cha.

\* **Cách hoạt động của pipe:**

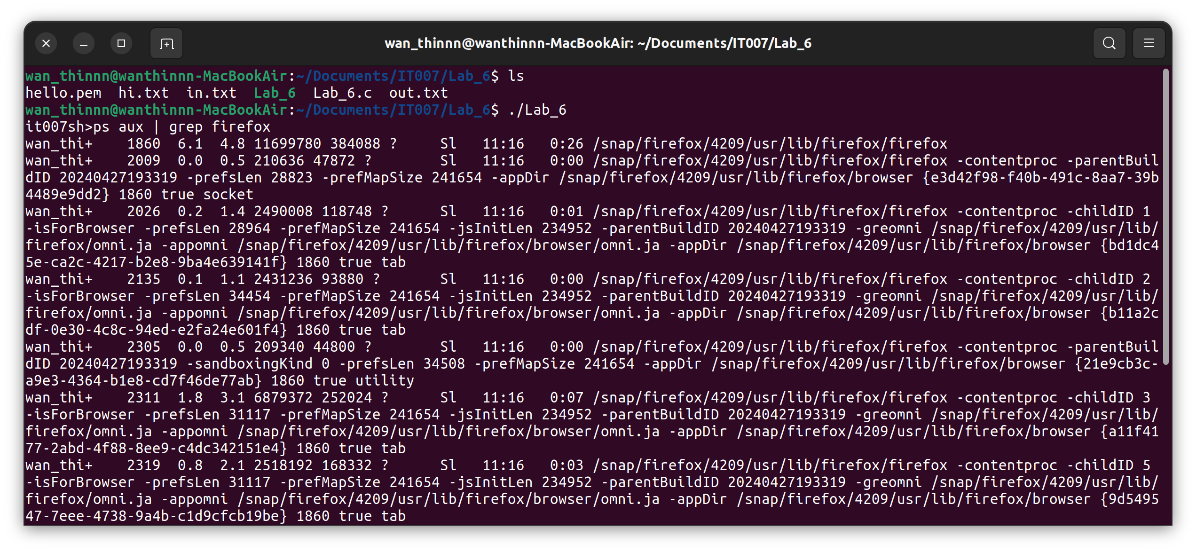
1. **Tạo pipe**: Hệ thống tạo ra hai file descriptors, một cho đầu đọc và một cho đầu ghi.
2. **Tạo tiến trình con thứ hai**: Tiến trình con thứ hai thiết lập để ghi đầu ra của lệnh đầu tiên vào đầu ghi của pipe.
3. **Thiết lập trong tiến trình cha**: Tiến trình cha thiết lập để đọc đầu vào từ đầu đọc của pipe.
4. **Ghi và đọc qua pipe**:
   * Khi lệnh đầu tiên (**ps aux**) ghi dữ liệu vào pipe, dữ liệu đó sẽ được lưu trữ tạm thời trong pipe.
   * Lệnh thứ hai (**grep firefox**) sẽ đọc dữ liệu từ pipe như thể nó đang đọc từ đầu vào chuẩn (**stdin**).

\* **Ví dụ hoạt động:**

Giả sử ta nhập lệnh **ps aux | grep firefox**:

* Lệnh **ps aux** sẽ được thực thi trong tiến trình con thứ hai.
* Đầu ra của **ps aux** sẽ được ghi vào pipe.
* Tiến trình cha đọc đầu vào từ pipe và thực thi lệnh **grep firefox**.
* Kết quả là **grep firefox** sẽ tìm kiếm từ khóa "firefox" trong đầu ra của **ps aux**.

**KẾT QUẢ:**



1. **Câu 5**

**CODE:**

**Ảnh có chứa văn bản, Phông chữ, màu trắng, đại số

Mô tả được tạo tự động**

* Khi người dùng nhấn Ctrl+C, hàm on\_sigint sẽ được gọi. Trong hàm này:
* In ra thông báo “Ctrl C was pressed”.
* Gửi tín hiệu SIGINT đến tiến trình con đang chạy ‘kill(pid, SIGINT);’.
* Ngoài ra đoạn được tô vàng nhằm khai báo ‘pid’ được khai báo để lưu trữ process ID của tiến trình con. Điều này giúp chúng ta có thể gửi tín hiệu để kết thúc tiến trình con khi cần.

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

Mô tả được tạo tự động

* Đoạn mã này đặt hàm on\_sigint làm hàm xử lý cho tín hiệu SIGINT (Ctrl+C).

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

Mô tả được tạo tự động

* Lệnh pid = fork() -> tạo một tiến trình con
* Nếu pid = 0, tức là đang ở trong tiến trình con, thực thi lệnh bằng execvp.
* Nếu pid > 0, tức là đang ở trong tiến trình cha, chờ tiến trình con kết thúc bằng wait(NULL);
* Vậy khi người dùng nhấn Ctrl+C:
* Tín hiệu SIGINT được gửi đến tiến trình cha.
* Hàm on\_sigint được gọi và gửi tiếp tín hiệu SIGINT đến tiến trình con.
* Tiến trình con nhận tín hiệu và kết thúc.
* Tiến trình cha thoát khỏi hàm wait(NULL); và hiển thị lại dấu nhắc it007sh>.

**KẾT QUẢ:**

