**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----



**BÀI TẬP LỚN MÔN HỌC NGUYÊN LÝ HỆ ĐIỀU HÀNH**

***ĐỀ TÀI***: Nghiên cứu tìm hiểu về quản lý tiến trình trong HĐH Linux.

**Giảng Viên** **:** Ths.Nguyễn Tuấn Tú

**Nhóm Số :** Nhóm 2

**Lớp : IT6025.6(006)K15**

HÀ NỘI, 2022

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----



**BÀI TẬP LỚN MÔN HỌC NGUYÊN LÝ HỆ ĐIỀU HÀNH**

***ĐỀ TÀI***: Nghiên cứu tìm hiểu về quản lý tiến trình trong HĐH Linux.

**Giảng Viên** **:** Ths.Nguyễn Tuấn Tú

**Sinh viên thực hiện :** Nguyễn Đình Hoàng

Trần Đình Hiệp

Nguyễn Huy Khánh

Đặng Kiều Trang

Lưu Đặng Trung Cheng

**Lớp : IT6025.6(006)K15**

HÀ NỘI, 2022

MỤC LỤC

Trang

[Chương 1: GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH LINUX 5](#_Toc102939439)

[1.1. Tổng quan về hệ điều hành 5](#_Toc102939440)

[1.1.1 Khái niệm hệ điều hành 5](#_Toc102939441)

[1.1.2 Các thành phần cơ bản của hệ điều hành 5](#_Toc102939442)

[1.1.3 Phân loại hệ điều hành 6](#_Toc102939443)

[1.2. Hệ điều hành linux 6](#_Toc102939444)

[Chương 2: KHÁI NIỆM VỀ TIẾN TRÌNH 9](#_Toc102939445)

[2.1. Khái niệm về tiến trình 9](#_Toc102939446)

[2.2. Tiến trình trong hệ điều hành linux 9](#_Toc102939447)

[Chương 3: CÁC TRẠNG THÁI CỦA MỘT TIẾN TRÌNH 11](#_Toc102939448)

[Chương 4: QUAN HỆ GIỮA CÁC TIẾN TRÌNH 12](#_Toc102939449)

[4.1. Tiến trình độc lập 12](#_Toc102939450)

[4.2. Tiến trình song song 12](#_Toc102939451)

[Chương 5: Quản lý tiến trình trên Linux 14](#_Toc102939452)

[5.1. Tại sao phải quản lý tiến trình 14](#_Toc102939453)

[5.2. Các loại tiến trình chính trên Linux 14](#_Toc102939454)

[5.3. Tạo một tiến trình 15](#_Toc102939455)

[5.4. Dừng một tiến trình 15](#_Toc102939456)

[5.5. Giao tiếp giữa các tiến trình 16](#_Toc102939457)

[5.5.1. Gửi tín hiệu đến tiến trình 16](#_Toc102939458)

[5.5.2. Nhận và xử lí tín hiệu : 18](#_Toc102939459)

[5.6. Liên lạc giữa 2 tiến trình 19](#_Toc102939460)

[5.6.1. Shared memory 19](#_Toc102939461)

[5.6.2. Message queue 22](#_Toc102939462)

[5.7. Lập lịch đa tiến trình 23](#_Toc102939463)

[5.8. Liên lạc giữa tiến trình cha và tiến trình con. 25](#_Toc102939464)

[5.9. Giám sát và điều khiển các tiến trình. 27](#_Toc102939465)

[KẾT LUẬN 31](#_Toc102939466)

Lời mở đầu

Những năm qua, Linux đã thực sự tạo ra một cuộc cách mạng trong lĩnh vực máy tính. Sự phát triển và những gì chúng mang lại cho máy tính thật đáng kinh ngạc: một hệ điều hành đa nhiệm, đa người dùng. Linux có thể chạy trên nhiều bộ vi xử lý khác nhau như: Intel , Motorola , MC68K , Dec Alpha. Nó tương tác tốt với các hệ điều hành: Apple , Microsoft và Novell. Không phải ngẫu nhiên mà ngành công nghệ thông tin Việt Nam chọn Linux làm hệ điều hành nền cho các chương trình ứng dụng chủ đạo về kinh tế và quốc phòng. Với mã nguồn mở, sử dụng Linux an toàn hơn các ứng dụng Windows. Linux đem đến cho chúng ta lợi ích về kinh tế với rất nhiều phần mềm miễn phí. Mã nguồn mở của hệ điều hành và của các chương trình trên Linux là tài liệu vô giá để chúng ta học hỏi về kỹ thuật lập trình vốn là những tài liệu không được công bố đối với các ứng dụng Windows. Trong đồ án này, chúng ta sẽ tìm hiểu một phần rất quan trọng trong hệ điều hành Linux đó là: quản lý tiến trình trong Linux. Một hệ điều hành muốn chạy ổn định thì phải có một cơ chế quản lý tiến trình hiệu quả. Cơ chế này sẽ được trình bày một cách chi tiết trong đồ án và có kèm theo các chương trình minh họa

Chương 1: GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH LINUX

1.1. Tổng quan về hệ điều hành

1.1.1 Khái niệm hệ điều hành

Hệ điều hành là một phần mềm hệ thống dùng để điều hành, quản lí các thiết bị phần cứng và các tài nguyên phần mềm trên máy tính. Hệ điều hành đóng vai trò trung gian trong việc giao tiếp giữa người sử dụng và phần cứng máy tính, cung cấp một môi trường cho phép người sử dụng phát triển và thực hiện ứng dụng của họ một cách dễ dàng. Khi khởi động máy tính, phần mềm hệ điều hành được khởi động đầu tiên, sau đó người sử dụng mới có thể sử dụng được các chương trình ứng dụng khác thông qua giao diện tương tác do hệ điều hành cung cấp.

**Nhiệm vụ cơ bản của hệ điều hành**

* Điều khiển vò quỏn lí trực tiếp các phần cứng.
* Thực hiện một số thao tác cơ bản trong máy tính như các thao tác đọc, viết tập tin, quản lí hệ thống tập tin và các kho dữ liệu.
* Cung cấp một hệ thống giao diện sơ khai cho các ứng dụng, thường thông qua một hệ thống thư viện các hàm chuẩn để điều hành các phần cứng mà từ đó các ứng dụng có thể gọi tới.
* Cung cấp một hệ thống lệnh cơ bản để điều hành máy. Các lện này gọi là lệnh hệ thống (system command).
* Hệ điều hành còn cung cấp một số phần mềm ứng dụng cơ bản như: trình duyệt web, chương trình soạn thảo văn bản, chương trình nghe nhạc, chương trình chỉnh sửa ảnh

1.1.2 Các thành phần cơ bản của hệ điều hành

* + **Hệ thống quản lí tiến trình:** Tạo và hủy các tiến trình; lập lịch thực hiện các tiến trình
  + **Hệ thống quản lí bộ nhớ chính:** Lưu giữ thông tin về các vị trí trong bộ nhớ; quyết định tiến trình nào được nạp vào bộ nhớ chính; cấp phát và thu hồi bộ nhớ khi cần thiết
  + **Hệ thống quản lí bộ nhớ phụ:** Quản lí các vùng trống trên đĩa, định vị, lưu trữ, lập lịch cho đĩa
  + **Hệ thống quản lí nhập xuất:** Cung cấp cơ chế làm việc thân thiện, dễ thao tác hơn, che dấu đi những đặc thù của phần cứng
  + **Hệ thống quản lí tập tin:** Hỗ trợ thao tác với tập tin như tạo, xóa, khôi phục, ánh xạ tập tin trên hệ thống phụ.
  + **Hệ thống bảo vệ:** Kiểm soát quá trình truy suất của chương trình, tiến trình hoặc người sử dụng với tài nguyên hệ thống
  + **Hệ thống dịch lệnh:** Đọc và thông dịch các lệnh điều khiển

1.1.3 Phân loại hệ điều hành

* + **Hệ điều hành đơn nhiệm một người sử dụng:**

Là hệ điều hành chỉ cho phép tại một thời điểm chỉ có một chương trình được thực thi, các chương trình phải thực hiện lần lượt

* + **Hệ điều hành đa nhiệm một người sử dụng:**

Cho phép một người đăng nhập vào hệ thống nhưng có thể kích hoạt cho hệ thống thực hiện nhiều chương trình

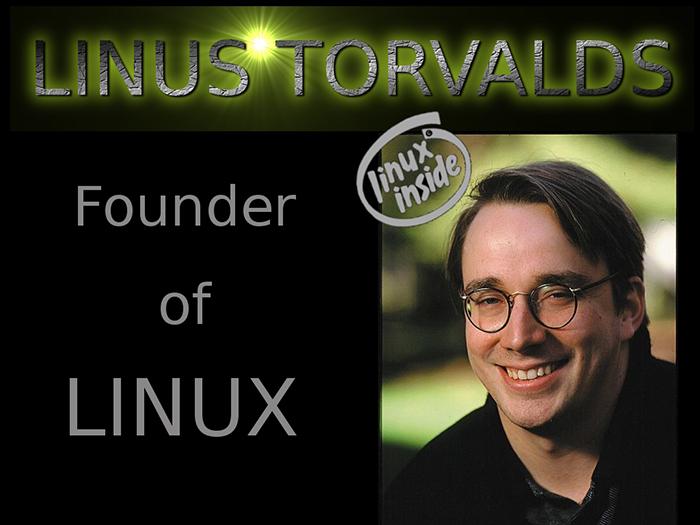
* + **Hệ điều hành đa nhiệm nhiều người sử dụng:**

Cho phép nhiều người đăng nhập vào hệ thống tại các thời điểm khác nhau và thực hiện đồng thời nhiều chương trình. Đòi hỏi máy tính phải có bộ xử lí mạnh và bộ nhớ trong lớn.

1.2. Hệ điều hành linux

* **LỊCH SỬ:**

Vào năm 1991, khi đang theo học tại đại học Helsinki, ông Linus Torvalds đã bắt đầu nảy sinh ý tưởng cho một hệ điều hành mới thay thế cho hệ điều hành cũ kỹ của nền giáo dục hiện tại. Chính vì vậy ông đã bắt tay vào viết những dòng lệnh đầu tiên của Linux, đặt nền móng cho sự phát triển mạnh mẽ của hệ điều hành Linux hiện nay.



*Hình 1.1 Linus Torvalds – Cha đẻ của hệ điều hành Linux*

Ngày nay, Linux được phân ra làm nhiều nhánh như: Ubuntu, Linux Mint, Fedora… nhưng thông dụng nhất hiện nay đang là Ubuntu.

**Ưu điểm**

* **Bản quyền:**

Hiện nay, ở Việt Nam nói riêng thì tỷ lệ người sử dụng Windows lậu đang chiếm con số rất cao.

Lợi thế của Linux chính là nền tảng mã nguồn mở và miễn phí. Nếu như sử dụng Windows bản quyền và bộ Microsoft Office bản quyền thì bạn sẽ phải chi khoảng vài triệu. Còn Linux thì không, bạn sẽ không phải bỏ xu nào mà vẫn sử dụng đầy đủ các tính năng, cũng như bộ ứng dụng văn phòng miễn phí như OpenOffice và LibreOffice.

* **Bảo mật:**

Nếu như trên Windows bạn luôn phải chật vật đối mặt với ngày càng nhiều những con virus, mã độc,… thì bạn lại được an toàn khi sử dụng Linux, bởi vì đơn giản, tất cả bọn chúng đều không thể hoạt động được trên nền tảng này. Công việc của bạn chỉ là xóa khi thấy bọn chúng trong USB hay ổ cứng di động.

* **Linh hoạt**

Trên Linux, nếu bạn có nhiều hiểu biết về nó, bạn có thể dễ dàng chỉnh sửa theo ý mình, còn trên Windows, nếu không có sự chấp thuận của Microsoft bạn sẽ không được phép làm điều đó. Hơn nữa, Linux còn mang lại sự tương thích với rất nhiều môi trường khác nhau và đây là một môi trường lý tưởng cho các lập trình viên cũng như các nhà phát triển.

* **Hoạt động mượt mà trên các máy tính có cấu hình yếu**

Khi Windows tung ra một bản nâng cấp phiên bản, kéo theo đó là sự nâng lên về yêu cầu phần cứng, khi đó nếu laptop có cấu hình không đủ, thì người dùng sẽ dừng lại ở phiên bản cũ đó và không còn được Microsoft “chăm lo” nữa hoặc sẽ phải nâng cấp cho phần cứng của mình. Còn đối với Linux thì không, hệ điều hành này hoạt động mượt mà và cực kỳ ổn định trên các máy tính có cấu hình thấp và vẫn được nâng cấp, hỗ trợ thường xuyên từ cộng đồng lập trình Linux.

**Nhược điểm**

Nói đi thì cũng phải nói lại, dù có những ưu điểm mạnh mẽ nhưng Linux vẫn tồn tại một số nhược điểm đáng buồn. Có lẽ vì những nhược điểm này mà Windows đang trở nên độc tôn.

* Số lượng ứng dụng hỗ trợ trên Linux còn rất hạn chế.
* Một số nhà sản xuất không phát triển driver hỗ trợ nền tảng Linux.
* Khó làm quen, đặc biệt nếu bạn đã quá quen thuộc với Windows thì khi chuyển sang Linux, bạn sẽ cần một khoảng thời gian để làm quen nó.

Chương 2: KHÁI NIỆM VỀ TIẾN TRÌNH

2.1. Khái niệm về tiến trình

Tiến trình - process là một khái niệm cơ bản trong bất kì một hệ điều hành nào. Một tiến trình có thể được định nghĩa là một thực thể chương trình đang được chạy trong hệ thống. Một web server chạy trong thiết bị là một tiến trình, hoặc một chương trình soạn thảo văn bản đang chạy trong thiết bị cũng là một tiến trình.

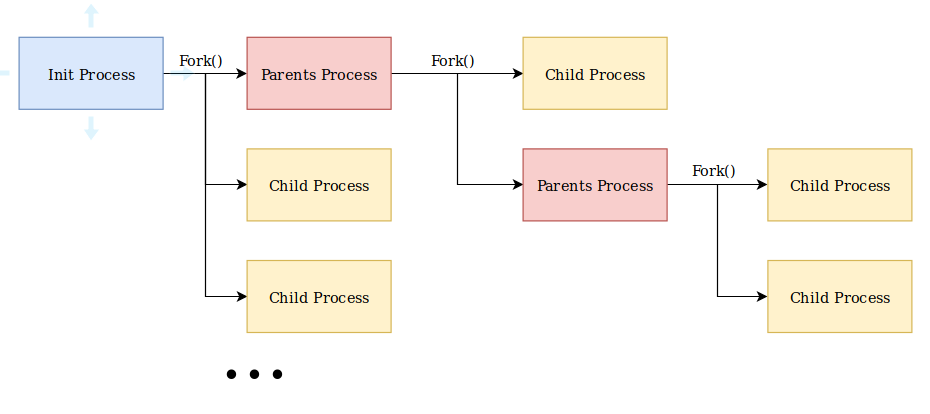
Một tiến trình cũng trải qua các quá trình như con người: Nó được sinh ra, nó có thể có một cuộc đời ít ý nghĩa hoặc nhiều ý nghĩa, nó có thể sinh ra một hoặc nhiều tiến trình con, và thậm chí, nó có có thể chết đi. Điều khác biệt nhỏ duy nhất là: tiến trình không có giới tính. Mỗi tiến trình chỉ có một tiến trình cha duy nhất.

Khi một tiến trình con được tạo ra, nó hầu như giống hệt như tiến trình cha. Tiến trình con sao chép toàn bộ không gian địa chỉ, thực thi cùng một mã nguồn như tiến trình cha, và bắt đầu chạy tiếp những mã nguồn riêng cho tiến trình con từ thời điểm gọi hàm tạo tiến trình mới.

Mặc dù tiến trình cha và tiến trình con cùng chia sẻ sử dụng phần mã nguồn của chương trình, nhưng chúng lại có phần dữ liệu tách biệt nhau (stack và heap). Điều này có nghĩa là, những sự thay đổi dữ liệu của tiến trình con không ảnh hưởng đến dữ liệu trong tiến trình cha.

2.2. Tiến trình trong hệ điều hành linux

Trong hệ điều hành linux các tiến trình được phân thành parents process và child process. Một tiến trình khi thực hiện lệnh fork() để tạo ra một tiến trình mới thì đưọc gọi là parents process. Tiến trình mới tạo được gọi là child process.

[](https://github.com/tronghuan98hd/thuctapsinh/blob/master/HuanTT/Process/images/process01.png)

*Hình 2.1 Sơ đồ tiến trình trong hệ điều hành linux*

Một parents process có thể có nhiều child process nhưng một child process chỉ có một parents process. Khi quan sát thông tin của một tiến trình, ngoài PID (Processes ID) ta cần để ý tới PPID (Parent Processes ID). Nó sẽ cho ta thông tin về parents process của tiến trình đó.

Chương 3: CÁC TRẠNG THÁI CỦA MỘT TIẾN TRÌNH

Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điếm đó. Trong quá trình sống một tiến trình thay đổi trạng thái do nhiều nguyên nhân như: phải chờ một sự kiện nào đó xảy ra ,hay đợi một thao tác nhập xuất hoàn tất, buộc phải dừng hoạt động do hết thời gian xử lý……

New

Ready

Running

Halt

Waiting

*Hình 3.1: Các trạng thái của một tiến trình*

**- New**(mới khởi tạo): Tiến trình mới được đưa vào hàng đợi. Hệ điều hành đã tạo ra các thông tin về tiến trình tuy nhiên tiến trình chưa được them và danh sách những tiến trình được phép thực hiện. Thông thường, tiến trình ở trạng thái này chưa nằm trong bộ nhớ.

**- Ready**(sẵn sàng): Chương trinh đưa vào hàng đợi ở trạng thái sẵn sàng được thi hành.

**- Waiting**(chờ đợi): Tiến trình được đưa vào trạng thái chờ.

**- Running**(chạy): Tiến trình được thi hành.

**- Halt**(kết thúc): Kết thúc tiến trình. Tiến trình không còn nằm trong danh sách các tiến trình được thực hiện nhưng vẫn chưa bị xoá. Tiến trình thuộc về trạng thái này sau khi đã thực hiện xong hoặc bị tiến trình khác kết thúc.

Hệ thống ghi nhận các trạng thái của tiến trình thông qua khối mô tả tiến trình.

Khối mô tả tiến trình bao gốm các thành phần:

+ Con trỏ trạng thái của tiến trình (Cho biết trạng thái hiện tại của tiến trình).

+ Vùng lưu trữ giá trị các thanh ghi mà tiến trình đang sử dụng.

+ Số thứ tự của tiến trình.

+ Thông tin về tài nguyên tiến trình đang sử dụng hoặc được phép sử dụng.

Chương 4: QUAN HỆ GIỮA CÁC TIẾN TRÌNH

Các tiến trình hoạt động trong hệ thống tồn tại hai mối quan hệ : Độc lập và song song.

4.1. Tiến trình độc lập

Tiến trình được gọi là độc lập nếu hoạt động của nó không ảnh hưởng hoặc không bị ảnh hưởng bởi các tiến trình khác đang hoạt động của hệ thống.

Tiến trình độc lập có những đặc trưng sau:

+ Trạng thái của nó không bị chia sẻ với bất kỳ tiến trình nào.

+ Việc thực hiện tiến trình là đơn định (Kết quả của tiến trình phụ thuộc vào đầu vào).

+ Tiến trình có thể được lặp lại.

+ Tiến trình có thể bị dừng hoặc bắt đầu lại mà không gây ảnh hưởng tới các tiến trình khác (Sự cố bất thường xảy ra trong hệ thống).

4.2. Tiến trình song song

Tiến trình được gọi là song song nếu hoạt động của nó ảnh hưởng tới các tiến trình khác đang hoạt động trong hệ thống.

Tiến trình song song được chia thành nhiều loại:

+ Tiến trình song song độc lập: Các tiến trình hoạt động song song nhưng không có quan hệ thông tin với nhau, trong trường hợp này hệ điều hành phải thiết lập cơ chế bảo vệ dữ liệu của các tiến trình, và cấp phát tài nguyên cho các tiến trình một cách hợp lý.

+ Tiến trình song song có quan hệ thông tin : Trong quá trình hoạt động các tiến trình trao đổi thông tin với nhau.Hai tiến trình A và B được gọi là có quan hệ thông tin với nhau nếu tiến trình này có gửi thông báo cho tiến trình kia.

+ Tiến trình song song phân cấp : Trong qua trình hoạt động một tiến trình có thể khởi tạo các tiến trình khác hoạt động song song với nó, tiến trình khởi tạo được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con.

+ Tiến trình song song đồng mức: Là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc lần lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

Tiến trình song song có những đặc trưng sau:

+ Trạng thái của nó bị chia sẻ cho các tiến trình khác.

+ Việc thực hiện tiến trình không đơn điệu (Kết quả của tiến trình phụ thuộc vào dãy thực hiện tương ứng và không dự báo trước).

+ Việc thực hiện tiến trình không đơn định (Kết quả của tiến trình không giống nhau với cùng một giá trị đầu vào).

Chương 5: Quản lý tiến trình trên Linux

5.1. Tại sao phải quản lý tiến trình

\* Nhiệm vụ của quản lý tiến trình :   
- Tạo lập, hủy bỏ tiến trình.   
- Tạm dừng, tái kích hoạt tiến trình.   
- Tạo cơ chế thông tin liên lạc giữa các tiến trình .  
- Tạo cơ chế đồng bộ hóa giữa các tiến trình.

\* Mục tiêu :   
- Hạn chế tối đa xung đột và bế tắc xảy ra, đưa ra giải pháp nếu xảy ra các tình huống đó.  
- Tận dụng tối đa khả năng của CPU (bài toán lập lịch).

Tiến trình là một môi trường thực hiện, bao gồm một phân đoạn lệnh và một phân đoạn dữ liệu. Cần phân biệt với khái niệm chương trình chỉ gồm tập hợp lệnh. Trên hệ điều hành Linux, tiến trình được nhận biết thông qua số hiệu của tiến trình, gọi là pid. Cũng như đối với user, nó có thể nằm trong nhóm. Vì thế để phân biệt ta nhận biết qua số hiệu nhóm gọi là PRGP. Một số hàm của C cho phép lấy được những thông số này:

int getpid() : trả về giá trị int là pid của tiến trình hiện tại

int getppid() : trả về giá trị int là pid của tiến trình cha của tiến trình hiện tại

int getpgrp() : trả về giá trị int là số hiệu của nhóm tiến trình

int setpgrp() : trả về giá trị int là số hiệu nhóm tiến trình mới tạo ra

Ví dụ:

Lệnh : printf("Toi la tien trinh %d thuoc nhom %d",getpid(),getgrp());

Kết quả sẽ là: Toi là tien trinh 235 thuoc nhom 231

5.2. Các loại tiến trình chính trên Linux

Tiến trình với đối thoại (Interactive processes): là tiến trình khởi động và quản lý bởi shell, kể cả tiến trình foreground hoặc background.

Tiến trình batch (Batch processes): Tiến trình không gắn liền đến bàn điều khiển (terminal) và được nằm trong hàng đợi để lần lượt thực hiện.

Tiến trình ẩn trên bộ nhớ Daemon processes. Daemon processes là các tiến trình chạy dưới background. Đa số các server cho các dịch vụ chạy theo phương thức này. Đây là các chương trình sau khi được gọi lên bộ nhớ, đợi thụ động các yêu cầu từ các client để trả lời sau các port xác định. Hầu hết các dịch vụ Internet như email, Web, DNS ... chạy theo nguyên tắc này. Các chương trình được gọi là các chương trình daemon và tên của nó thường kết thúc bằng ký tự “d” như named, inetd ...

5.3. Tạo một tiến trình

int fork() tạo ra một tiến trình con. Giá trị trả lại là 0 cho tiến trình con và dấu hiệu pid cho tiến trình cha. Giá trị sẽ là -1 nếu không tạo được tiến trình mới. Theo nguyên tắc cơ bản của hệ thống, tiến trình con và cha sẽ có cùng đoạn mã. Đoạn dữ liệu của tiến trình mới là một bản sao chép chính xác đoạn dữ liệu của tiến trình cha. Tuy nhiên tiến trình con vẫn khác tiến trình cha ở pid, thời gian xử lý, ...

5.4. Dừng một tiến trình

Lệnh kill thường được sử dụng để ngừng thi hành một tiến trình dựa trên định danh của tiến trình PID :  
Kill [signal] <PID>

Signal : là một số hay tên của tín hiệu được gửi tới tiến trình.

PID : mã số nhận diện tiến trình muốn dừng.

Lệnh killall: dùng để kết thúc tất cả các tiến trình của một câu lệnh thông qua việc truyền tên của câu lệnh dưới dạng một tham số:

Killall tên\_lệnh

Người dùng thông thường có thể chấm dứt các tiến trình của riêng họ, nhưng không phải các tiến trình thuộc về người dùng khác

Lệnh kill có thể gởi bất kỳ tín hiệu signal nào tới một tiến trình, nhưng theo mặc định nó gửi tín hiệu 15, TERM (là tín hiệu kết thúc chương trình).

Các signal thường dùng lệnh kill bao gồm:

SIGHUP 1 - Hangup (gọi lại tiến trình).

SIGINT 2 - Ngắt từ bàn phím (Ctrl+C).

SIGKILL 9 - Hủy tiến trình.

SIGTERM 15 - Kết thúc tiến trình.

SIGSTOP 17, 19, 23: - Dừng tiến trình.

Khi kết thúc một tiến trình hay một chuỗi các tiến trình, thông thường nên tiến hành thử với tín hiệu ít gây nguy hiểm nhất, SIGTERM, nếu không được mới sử dụng các tín hiệu INT hay KILL.

5.5. Giao tiếp giữa các tiến trình

Việc giao tiếp giữa các tiến trình được thực hiện thông qua các tín hiệu chuẩn của hệ thống. Chúng được sử dụng để báo hiệu các sự kiện không đồng bộ cho một hoặc nhiều tiến trình. Mỗi tín hiệu có thể kết hợp hoặc có sẵn bộ xử lý tín hiệu (signal handler). Tín hiệu sẽ ngắt ngang quá trình xử lý của tiến trình, bắt hệ thống chuyển sang gọi bộ xử lý tín hiệu ngay tức khắc. Khi kết thúc xử lý tín hiệu, tiến trình lại tiếp tục thực thi.

5.5.1. Gửi tín hiệu đến tiến trình

a. Các nguồn gửi signals:

\* Từ phần cứng (ví dụ lỗi do các phép tính số học)

\* Từ kernel:

Khi xảy ra một số điều kiện về phần cứng (SIGSEGV, SIGFPE)

Khi xảy ra điều kiện phần mềm (SIGIO)

\* Từ người dùng đầu cuối

\* Từ một tiến trình gửi đến một tiến trình khác ( ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc)

b. Các cách gửi tín hiệu đến tiến trình:

\* Từ bàn phím

**Ctrl+C**: gửi tín hiệu **INT( SIGINT )** đến tiến trình, ngắt ngay tiến trình (interrupt).

**Ctrl+Z**: gửi tín hiệu **TSTP( SIGTSTP )** đến tiến trình, dừng tiến trình (suspend).

**Ctrl+/**: gửi tín hiệu **ABRT( SIGABRT )** đến tiến trình, kết thúc ngay tiến trình (abort).

\* Từ dòng lệnh

Lệnh kill thường được sử dụng để ngừng thi hành một tiến trình. Lệnh kill có thể gởi bất kỳ tín hiệu signal nào tới một tiến trình, nhưng theo mặc định nó gởi tín hiệu 15, TERM (là tín hiệu kết thúc chương trình).

kill -<signal> <PID>

Ví dụ: kill -INT 2309 hoặc kill -2 2309 dùng gửi tín hiệu **INT** ngắt tiến trình có **PID 2309**.

Nếu không chỉ định tên tín hiệu, tín hiệu **TERM** được gửi để kết thúc tiến trình.

Lệnh **fg**: gửi tín hiệu **CONT** đến tiến trình, dùng đánh thức các tiến trình tạm dừng do tín hiệu **TSTP** trước đó.

Bằng các hàm hệ thống kill():

Ví dụ :

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int main(void){

pid\_t retVal;

retVal = fork();

if(retVal > 0){

int i = 0;

while(i++ < 5){

printf("in the parent process.\n");

sleep(1);

}

//hủy tiến trình con

kill(retVal, SIGKILL);

} else if (retVal == 0){

int i = 0;

//Không vượt quá 15, vì tiến trình cha sẽ hủy nó

while(i++ < 15){

printf("In the child process.\n");

sleep(1);

}

} else {

printf("Something bad happened.");

exit(EXIT\_FAILURE);

}

return 0;

}

5.5.2. Nhận và xử lí tín hiệu :

a. Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau :

- Bỏ qua tín hiệu

- Xử lý tín hiệu theo kiểu mặc định

- Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

b. Bộ xử lý tín hiệu mặc định

Hệ thống đã dành sẵn các hàm mặc định xử lý tín hiệu cho mỗi tiến trình.

Ví dụ:

Bộ xử lý mặc định cho tín hiệu **TERM** gọi là hàm **exit()** - chấm dứt tiến trình hiện hành.

Bộ xử lý dành cho tín hiệu **ABRT** là gọi hàm hệ thống **abort()** để tạo ra file core lưu xuống thư mục hiện hành và thoát chương trình.

Mặc dù vậy đối với một số tín hiệu bạn có thể cài đặt hàm thay thế bộ xử lý tín hiệu mặc định của hệ thống.

c. Cài đặt bộ xử lý tín hiệu :

Có nhiều cách thiết lập bộ xử lý tín hiệu (signal handler) thay cho bộ xử lý tín hiệu mặc định. Một cách cơ bản nhất đó là ta sẽ gọi hàm signal() hoặc sigaction().

Ví dụ:

#include <signal.h>

typedef void (\*sighandler\_t) (int);

sighandler\_t signal(int signum, sighandler\_t handler);

Hàm signal() sẽ gọi bộ xử lý được xác định với signum. Bộ xử lý có thể là **SIG\_IGN** (Bỏ qua tín hiệu), **SIG\_DFL** (Đặt tín hiệu trở lại cơ chế mặc định) hoặc bộ xử lý do người dùng xây dựng hay là 1 function address.

Ngoài ra Linux còn hỗ trợ một cách dùng signal mới hơn là sigaction:

#include <signal.h>

int sigaction(int signum, const struct sigaction \*act, struct sigaction \*oldact)

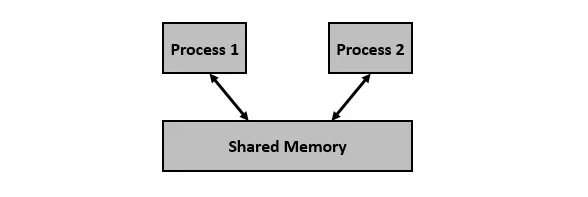
5.6. Liên lạc giữa 2 tiến trình

5.6.1. Shared memory

a. Khái niệm

*Shared memory (bộ nhớ được chia sẻ)* là cơ chế giao tiếp liên tiến trình (IPC) có sẵn trong Linux và các hệ thống giống Unix khác, khi một bộ nhớ chung được sử dụng cho hai hoặc nhiều tiến trình khác nhau.

Trong các cơ chế giao tiếp giữa các tiến trình khác như các [pipe (đường ống)](https://viblo.asia/p/giao-tiep-giua-cac-tien-trinh-trong-linux-phan-1-su-dung-signal-va-pipe-Qpmlejxr5rd) hay message queue (hàng đợi tin nhắn), cần thực hiện các bước gửi dữ liệu từ tiến trình này sang trình khác. Tuy nhiên, đối với shared memory, không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây cả, các tiến trình đều có thể truy cập vào bộ nhớ chung. Và giao tiếp được thực hiện thông qua bộ nhớ được chia sẻ này, nơi các thay đổi được thực hiện bởi một tiến trình có thể được xem bởi tiến trình khác.



*Hình 5.1 Quan hệ giữa tiến trình và bộ nhớ được chia sẻ*

Ở đây, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.

b. Sử dụng shared memory

Linux cung cấp các hàm hệ thống được sử dụng để tạo và sử dụng shared memory như sau:

1. shmget ()

int shmget(key\_t key, size\_t size, int shmflg)

Trong đó, các đối số:

key: là khóa giúp nhận ra một bộ nhớ được chia sẻ, nó có thể là một giá trị tùy ý hoặc một giá trị được tạo từ hàm ftok () - hàm giúp tạo một khóa duy nhất.

size: là kích thước của phân đoạn bộ nhớ được chia sẻ.

shmflg: chỉ định một cờ (flag/s) bắt buộc cho bộ nhớ được chia sẻ như IPC\_CREAT (tạo phân đoạn bộ nhớ mới) hoặc IPC\_EXCL (Được sử dụng với IPC\_CREAT để tạo phân đoạn bộ nhớ mới và lời gọi hàm sẽ không thành công, nếu phân đoạn này đã tồn tại). Lưu ý là đối số này còn cần đi kèm với các quyền truy cập vào bộ nhớ chia sẻ, bạn cần đặt các quyền phù hợp cho nhu cầu sử dụng của mình.

Sau khi lệnh gọi sẽ hàm này được thực hiện thành công, shmget () trả về một mã định danh cho phân đoạn bộ nhớ được chia sẻ.

2. shmat ()

void \* shmat(int shmid, const void \*shmaddr, int shmflg)

Trong đó, các đối số:

shmid: là định danh của phân đoạn bộ nhớ chia sẻ và chính là giá trị trả về của lệnh gọi hệ thống shmget ().

shmadrr: là chỉ định địa chỉ để gắn phân đoạn bộ nhớ. Thường chúng ta sẽ đặt nó là NULL và khi đó, mặc định hệ thống sẽ chọn địa chỉ phù hợp để gắn phân đoạn.

shmflg: chỉ định một cờ (flag/s) bắt buộc cho bộ nhớ được chia sẻ chẳng hạn như SHM\_RND (làm tròn địa chỉ thành SHMLBA) hoặc SHM\_EXEC (cho phép nội dung của phân đoạn được thực thi) hoặc SHM\_RDONLY (đính kèm phân đoạn với mục đích chỉ đọc, theo mặc định nó là đọc-ghi) hoặc SHM\_REMAP (thay thế ánh xạ hiện có trong phạm vi được chỉ định bởi shmaddr và tiếp tục cho đến khi kết thúc phân đoạn).

Sau khi lệnh gọi hàm này được thực hiện thanh công, nó sẽ trả về địa chỉ mà phân đoạn bộ nhớ chia sẻ được gắn

3. shmdt ()

int shmdt(const void \*shmaddr)

Sau khi tiến trình của bạn được hoàn thành với việc sử dụng bộ nhớ chia sẻ thì bạn có sẽ cần tách nó ra khỏi bộ nhớ chia sẻ. Điều này được thực hiện bằng cách gọi hàm hệ thống shmdt(). shmaddr là địa chỉ của phân đoạn bộ nhớ chia sẻ được tách ra.

4. shmctl ()

int shmctl(int shmid, int cmd, struct shmid\_ds \*buf)

Lệnh gọi hệ thống trên giúp thực hiện thao tác điều khiển cho một phân đoạn bộ nhớ chia sẻ với:

shmid: là ID định danh cho bộ nhớ chia sẻ.

cmd: là chỉ định một lệnh được sử dụng trên bộ nhớ được chia sẻ bao gồm:

IPC\_STAT - Sao chép thông tin về các giá trị hiện tại của từng thành phần trong cấu trúc struct shmid\_ds vào cấu trúc được chỉ ra bởi con trỏ buf. Lệnh này yêu cầu quyền đọc đối với phân đoạn bộ nhớ được chia sẻ.

IPC\_SET - Đặt ID người dùng, ID nhóm của chủ sở hữu, quyền, v.v. được trỏ đến theo cấu trúc buf.

IPC\_RMID - Đánh dấu phân đoạn sẽ bị hủy. Phân đoạn chỉ bị phá hủy sau khi tiến trình cuối cùng đã tách nó ra.

IPC\_INFO - Trả về thông tin về giới hạn bộ nhớ dùng chung và các tham số trong cấu trúc được trỏ bởi buf.

SHM\_INFO - Trả về cấu trúc shm\_info chứa thông tin về tài nguyên hệ thống được tiêu thụ bởi bộ nhớ được chia sẻ.

buf: là một con trỏ đến cấu trúc bộ nhớ dùng chung có tên struct shmid\_ds. Các giá trị của cấu trúc này sẽ được sử dụng cho cả set hoặc get theo cmd.

Tất cả các lệnh gọi hàm hệ thống trên, nếu xảy ra lỗi và không thành công sẽ trả về cho chúng ta giá trị -1.

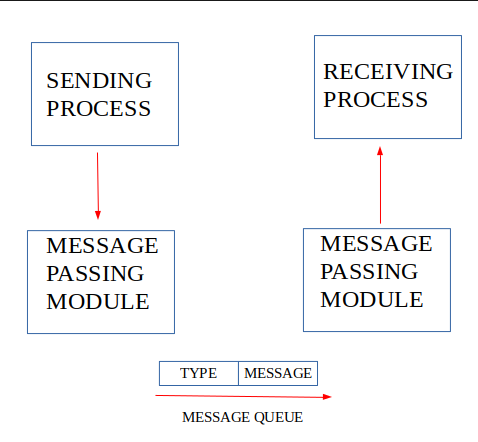
c. Đánh giá việc sử dụng shared memory

Ưu điểm của việc sử dụng shared memory là việc không cần đến truyền gửi dữ liệu giúp nó tiết kiệm được chi phí và là phương pháp nhanh nhất giúp trao đổi giữa các tiến trình.

Nhược điểm của phương pháp này chính là nó sẽ gây ra những khó khăn nhất định trong việc bảo đảm sự toàn vẹn dữ liệu (coherence) , ví dụ : làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi ? Làm thế nào ngăn cản hai tiến trình cùng đồng thời ghi dữ liệu vào vùng nhớ chung ?…Rõ ràng vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp.. Bên cạnh đó, shared memory cũng không phải là lựa chọn phù hợp trong các hệ phân tán , để trao đổi thông tin giữa các máy tính khác nhau.

5.6.2. Message queue

a. Khái niệm

*Message queue* hay *hàng đợi tin nhắn* cũng là một cơ chế IPC có sẵn trong Linux. Mỗi một khối dữ liệu được truyền đi được xác định một kiểu (TYPE) cụ thể và người nhận có thể nhận được các dữ liệu đó tùy theo kiểu của dữ liệu. Trong nhiều trường hợp sử dụng, điều này đem lại nhiều hiệu quả hơn thay vì phải nhận dữ liệu theo cách FIFO như cách sử dụng các pipe (đường ống)

*Hình 5.2 Minh họa sử dụng message queue*

b. Sử dụng message queue

Tương tự như với shared memory, để hỗ trợ cơ chế giao tiếp tiến trình bằng message queue, hệ điều hành cũng cung cấp các hàm IPC chuẩn (Interprocess communication) để thực hiện giao tiếp tiến trình với message queue, cơ bản là các hàm:

1. msgget (): trả về ID định danh cho message queue mới được tạo hoặc trả về định danh của một message queue đã tồn tại cùng với một giá trị khóa nhận dạng của nó.

int msgget(key\_t key, int msgflg)

2. msgsnd (): Hàm hệ thống này được sử dụng để đưa dữ liệu vào message queue.

int msgsnd(int msgid, const void \*msgp, size\_t msgsz, int msgflg)

3. msgrcv (): Hàm hệ thống thống này giúp lấy dữ liệu ra khỏi môtj message queue.

int msgrcv(int msgid, const void \*msgp, size\_t msgsz, long msgtype, int msgflg)

4. msgctl (): Hàm hệ thống này cũng giúp thực hiện thao tác điều khiển một message queue.

int msgctl(int msgid, int cmd, struct msqid\_ds \*buf)

c. Đánh giá việc sử dụng message queue

Sử dụng message queue mang lại nhiều đặc điểm khác so với sử dụng shared memory hay các phương pháp IPC khác như:

Đơn vị truyền thông tin trong cơ chế giao thiếp này là một data block(hay một thông điệp) và có xác định kiểu, do đó các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc.

Khi sử dụng shared memory, dữ liệu có sẵn cho nhiều tiến trình truy cập. Tuy nhiên, khi sử dụng message queue, khi một tiến trình đã nhận được dữ liệu, nó sẽ không còn được sử dụng cho bất kỳ tiến trình nào khác.

Không giống như pipe, message queue dựa trên thông điệp, trong khi đường ống dựa trên luồng byte và message queue không nhất thiết phải đọc trước ra trước.

Có một nhược điểm của message queue là độ dài tối đa của mỗi thông điệp bị giới hạn và vòng đời của các message queue được gắn liền với kernel.

5.7. Lập lịch đa tiến trình

Lập lịch đa tiến trình là về cơ bản, bạn sẽ có 1 tiến trình chính (main process), sau đó tiến trình này sẽ đẻ ra các tiến trình con để làm việc. Nhiệm vụ của tiến trình chính khá là đơn giản, không thể có lỗi, thường là nhiệm vụ quản lý và giao việc cho các tiến trình con hơn là trực tiếp xử lý một công việc gì đó

Ngắn gọn lại:

Tạo ra tiến trình chính, tiến trình chính gọi các tiến trình con

Tiến trình con nhận nhiệm vụ (thường là nhận nhiệm vụ thông qua giao tiếp vs tiến trình chính), xử lý công việc liên tục

Tiến trình chính quản lý tiến trình con, nếu thằng con chết/treo/hoàn thành nhiệm vụ thì xử lý (chết thì tạo mới, treo thì stop, ....)

Để liên lạc giữa các tiến trình thì chúng ta dùng các Ống dẫn liên lạc. Đó là một cơ chế cơ bản để liên lạc gián tiếp giữa các tiến trình , là các file đặc biệt (FIFO), ở đó các thông tin được truyền đi 1 đầu và thoát ra ở một đầu khác.

Các ống dẫn không cung cấp một truyền thông theo cấu trúc. Thao tác đọc độc lập với thao tác ghi và do đó tại cấp độ của các lệnh gọi hệ thống, người ta không thể biết được kích cỡ, người gởi và người nhận dữ liệu chứa trong pipe. Mặt khác, một ưu điểm của phương pháp truyền thông này là khả năng sử dụng dữ liệu đã ghi nhiều lần để được đọc chỉ một lần

Ống dẫn có các đặc điểm

- Các ống dẫn chỉ mang tính chất tạm thời, chỉ tồn tại trong thời gian thực hiện của một tiến trình tạo ra nó.

- Muốn tạo ra một ống dẫn phải bắt đầu bằng một lệnh đặc biệt: pipe().Hệ thống cung cấp cho ta hàm pipe() để tạo đường ống có khả năng đọc/ghi.

- Nhiều tiến trình có thể viết và đọc trên cùng một ống dẫn. Tuy nhiên, không có một cơ chế nào để phân biệt thông tin cho các tiến trình ở đầu ra.

- Dung lượng ống dẫn bị hạn chế (khoảng 4KB). Do đó khi chúng ta cố gắng viết khi ống dẫn bị đầy thì sẽ gặp phải trường hợp tắc nghẽn.

- Các tiến trình liên lạc qua ống dẫn phải có mối quan hệ họ hàng và các ống dẫn nối phải được mở trước khi tạo ra các tiến trình con.

-Không thể tự thay đổi vị trí thông tin trong ống.

Tạo một ống dẫn:

int p\_desc[2];

int pipe(p\_desc);

Giá trị trả về là 0 nếu thành công, -1 nếu thất bại.

p\_desc[0] : chứa các số hiệu mô tả nhờ đó có thể đọc trong ống dẫn.

p\_desc[1] : chứa các số hiệu mô tả nhờ đó có thể viết trong ống dẫn.

Như vậy việc viết trong p\_desc[1] là để truyền dữ liệu trong ống và việc đọc trong p\_desc[0] để nhận chúng.

Ví dụ:

*#include*

*#include*

*main()*

*{*

*int i,ret, p\_desc[2];*

*char c;*

*pipe(p\_desc);*

*write(p\_desc[1], "AB", 2);*

*for (i=1; i<=3,i ++) {*

*ret=read(p\_desc[0], &c, 1);*

*if (ret == 1)*

*printf(" Gia tri: %c\n",c);*

*else*

*perror("Loi ong dan rong");*

*}*

*}*

Ví dụ trên chỉ ra rằng ta có thể truyền và nhận thông tin trên ống dẫn. Hệ thống cung cấp cho ta hàm pipe() để tạo đường ống có khả năng đọc/ghi. Chúng ta đã dùng hàm read() và write() để viết (truyền) và đọc (nhận) trên ống dẫn.

5.8. Liên lạc giữa tiến trình cha và tiến trình con.

Như ta đã biết, khi nhân đôi tiến trình, bộ mô tả file của tiến trình cha và tiến trình con có khả năng kế thừa nhau cho nên ống dẫn mở bởi tiến trình cha cũng được nhân bản và sao chép sang tiến trình con. Tiến trình cha sẽ đọc dữ liệu nhập vào từ phía người dùng và ghi vào đường ống trong khi tiến trình con phía bên kia đường ống tiếp nhận dữ liệu bằng cách đọc từ đường ống và in ra màn hình.

Trong ví dụ dưới đây, một tiến trình tạo ra một ống dẫn, tạo ra một tiến trình con, viết một văn bản vào ống dẫn.Tiến trình con thừa hưởng ống dẫn và các ký hiệu mô tả của ống dẫn, thực hiện đọc trong ống dẫn:

*#include*

*#include*

*void code\_fils(int number) {*

*int fd, nread;*

*char texte[100];*

*- 31-*

*fd=number;*

*printf(" So hieu mo ta la %d\n",fd);*

*switch (nread=read(fd, texte, sizeof(texte)))*

*{*

*case -1:*

*perror("Loi doc.");*

*case 0:*

*perror("EOF");*

*default:*

*printf("Van ban nhan duoc co %d ky tu: %s\n",fd, texte);*

*}*

*}*

*main() {*

*int fd[2];*

*char chaine[10];*

*if (pipe(fd)==-1)*

*{ perror("Loi khoi tao pipe.");*

*exit(1);*

*}*

*switch (fork()) {*

*case -1:*

*perror(" Loi khoi tao tien trinh.");*

*break;*

*case 0:*

*if (close(fd[1])==-1)*

*perror(" Error.");*

*code\_fils(fd[0]);*

*exit(0);*

*}*

*close(fd[0]);*

*if (write(fd[1]),"hello",6)==-1)* *perror("Loi truyen.");*

*}*

Kết quả chương trình:

So hieu mo ta la: 5

Van ban nhan duoc co 6 ky tu: hello

Chú ý rằng, tiến trình con đọc trong ống dẫn mà không viết ở đó nên nó bắt đầu bằng cách đóng phần viết fd[1] để tiết kiệm các tín hiệu mô tả của tổ hợp. Tương tự, vì tiến trình cha chỉ sử dụng phần viết nên nó đóng phần đọc lại (fd[0]). Sau đó tiến trình cha viết vào ống dẫn 6 ký tự và tiến trình con đã đọc chúng.

Chương trình trên thực hiện cơ chế trao đổi đường ống một chiều. Tiến trình cha chuyển dữ liệu đến tiến trình con bằng cách ghi vào đường ống. Tiến trình con ngược lại đọc dữ liệu ra để xử lý ở đầu bên kia đường ống

5.9. Giám sát và điều khiển các tiến trình.

Một khối điều khiển tiến trình (Process Control Block - PCB) là một cấu trúc dữ liệu trong nhân hệ điều hành chứa thông tin cần thiết để quản lý một tiến trình nhất định.

Tuỳ thuộc vào cài đặt nhưng nói chung PCB trực tiếp hoặc gián tiếp chứa những thông tin sau:

- Định danh của tiến trình (process identifier hay PID).

- Giá trị các thanh ghi của tiến trình, trong đó đáng chú ý là con trỏ chương trình và con trỏ stack

- Không gian địa chỉ của tiến trình

- Độ ưu tiên (trong đó tiến trình có giá trị cao hơn được ưu tiên trước, ví dụ nice trong các hệ điều hành Unix)

- Thông tin kế toán tiến trình, ví dụ như thời điểm thực thi gần nhất, bao nhiêu thời gian CPU đã sử dụng...

- Con trỏ tới PCB tiếp theo, nghĩa là con trỏ tới tiến trình tiếp theo được chạy

- Thông tin V/R (ví dụ các thiết bị V/R được cấp phát cho tiến trình, danh sách các tệp đang mở...)

Lấy thông tin trạng thái của các tiến trình: sử dụng câu lệnh ps, pstree, top.

PS:

*#ps <option>*

Option:

f: thể hiện các process dưới dạng tree.

l: thể hiện dưới dạng long list.

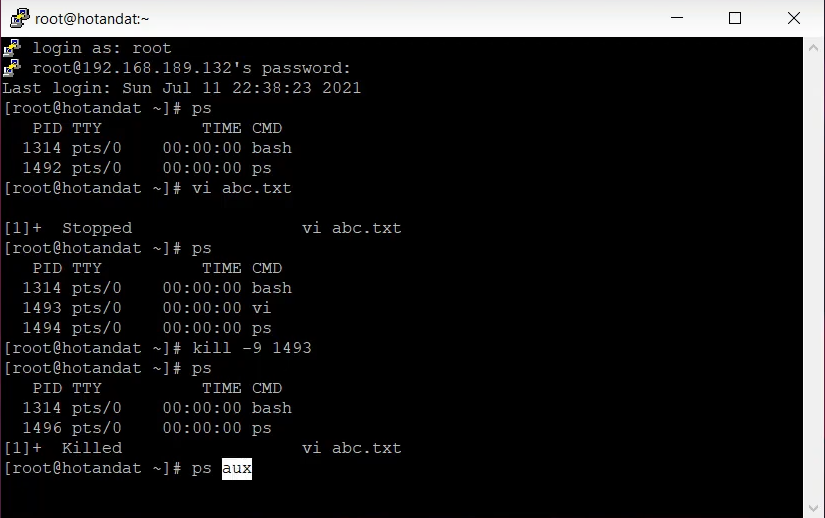
w: thể hiện dưới dạng wide output.

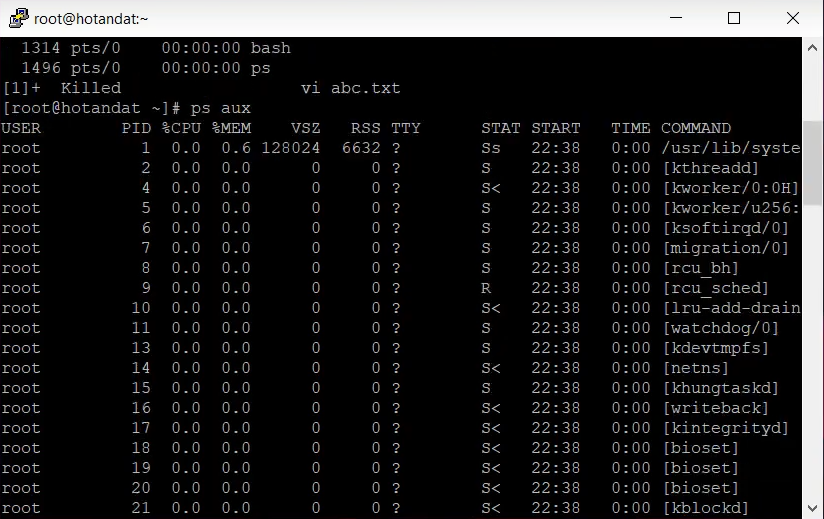
x: Xem cả các process không gắn với terminal (daemon).

a: process của các user khác.

U: user xem process của một user cụ thể.

u: thể hiện dưới dạng “user format”.





*Hình 5.3 và 5.4 Bảng quản lý và giám định tiến trình*

Trong đó:

|  |  |
| --- | --- |
| USER hoặc UID | Tên của tiến trình. |
| PID | ID (định danh) của tiến trình. |
| %CPU | %CPU sử dụng của tiến trình. |
| %MEM | % bộ nhớ tiến trình sử dụng. |
| SIZE | Kích thước bộ nhớ ảo tiến trình sử dụng. |
| RSS | Kích thước của bộ nhớ thực sử dụng tiến trình. |
| TTY | Vùng làm việc của tiến trình. |
| STAT | Trạng thái của tiến trình. |
| START | Thời gian hay ngày bắt đầu của tiến trình. |
| TIME | Tổng thời gian sử dụng CPU. |
| COMMAND | Câu lệnh được thực hiện. |
| PRI | Mức ưu tiên của tiến trình. |
| PPID | ID của tiến trình cha. |
| WCHAN | Tên của hàm nhận khi tiến trình ngủ được lấy từ file/boot/system.map |

Pstree:

Tương tự lệnh ps với tham số -f

Tham số -p in ra màn hình cà process ID

KẾT LUẬN

Qua phần bài tập được trình bày trên tài liệu này, ta đã nắm được các nguyên lí trong cơ chế quản lý tiến trình của hệ điều hành Linux ở một mức độ căn bản. Từ đó giúp hiểu được một phần cách vận hành hoạt động hệ thống của hệ điều hành Linux. Tài liệu đã mang đến 1 phần kiến thức tuy nhỏ nhưng khá đầy đủ và căn bản có thể giúp em tự tin khi nghiên cứu sâu hơn trong vấn đề quản lý bộ nhớ không chỉ của hệ điều hành Linux mà còn cả các hệ điều hành khác.

Tài liệu này chủ yếu nghiên cứu về quản lí tiến trình, một kỹ thuật quản lý bộ nhớ rất hiệu quả và thông dụng không chỉ áp dụng riêng với Linux. Phần bài tập này chỉ mang tính chất nghiên cứu cơ bản nên không trình bày sâu nhưng chúng em đã cố gắng tóm lược lại các vấn đề chính để hiểu được cách hoạt động của bộ nhớ trong hệ thống sử dụng hệ điều hành Linux

Sau bài tập lớn này chúng em đã có thêm nhiều hiểu biết về hệ điều hành Linux đặc biệt là quy trình quản lý tiến trình trong hệ điều hành Linux, giúp chúng em nâng cao khả năng đọc dịch tài liệu, phân kế hoạch thực hiện cũng như sắp xếp thời gian. Em xin cám ơn thầy và các bạn đã giúp đỡ chúng em hoàn thành bài tập lớn này!