**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**TIỂU LUẬN MÔN HỌC**

**HỆ ĐIỀU HÀNH**

**LỊCH SỬ RA ĐỜI, CÁC GIAI ĐOẠN PHÁT TRIỂN CỦA**

**HỆ ĐIỀU HÀNH LINUX**

**VÀ**

**CÁC VẤN ĐỀ LIÊN QUAN ĐẾN BÀI TOÁN PHÂN PHỐI**

**TÀI NGUYÊN CPU CHO CÁC TIẾN TRÌNH**

Thuộc nhóm ngành khoa học: Khoa học tự nhiên

**TP Hồ Chí Minh, 05/2023**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**TIỂU LUẬN MÔN HỌC**

**HỆ ĐIỀU HÀNH**

**LỊCH SỬ RA ĐỜI, CÁC GIAI ĐOẠN PHÁT TRIỂN CỦA**

**HỆ ĐIỀU HÀNH LINUX**

**VÀ**

**CÁC VẤN ĐỀ LIÊN QUAN ĐẾN BÀI TOÁN PHÂN PHỐI**

**TÀI NGUYÊN CPU CHO CÁC TIẾN TRÌNH**

Thuộc nhóm ngành khoa học: Khoa học tự nhiên

Nhóm SV thực hiện: Phạm Hoài Nhân - 47.01.104.147

Thái Gia Bảo - 47.01.104.049

Mai Thành Danh - 47.01.104.059

Trần Duy Quân - 47.01.104.169

Lớp, Khoa: K47.CNTT.C, Công nghệ Thông tin

Ngành học: Công nghệ thông tin

**TP. HỒ CHÍ MINH, 05/2023**

MỤC LỤC

[MỤC LỤC 3](#_Toc136014517)

[LỜI CÁM ƠN 5](#_Toc136014518)

[DANH MỤC CÁC HÌNH VẼ 6](#_Toc136014519)

[DANH MỤC CÁC BẢNG 7](#_Toc136014520)

[MỞ ĐẦU 8](#_Toc136014521)

[CHƯƠNG 1. LỊCH SỬ RA ĐỜI VÀ CÁC GIAI ĐOẠN PHÁT TRIỂN CỦA HỆ ĐIỀU HÀNH LINUX 11](#_Toc136014522)

[1.1 Lịch sử ra đời của Linux 11](#_Toc136014523)

[1.1.1 Unix - tiền thân của Linux 11](#_Toc136014524)

[1.1.2 Linux 14](#_Toc136014525)

[1.2 Các giai đoạn phát triển của hệ điều hành 16](#_Toc136014526)

[CHƯƠNG 2. CÁC VẤN ĐỀ LIÊN QUAN ĐẾN BÀI TOÁN PHÂN PHỐI TÀI NGUYÊN CPU CHO CÁC TIẾN TRÌNH 18](#_Toc136014527)

[2.1 Khái quát về bài toán 18](#_Toc136014528)

[2.2 Các thuật toán phân phối tài nguyên CPU 19](#_Toc136014529)

[2.2.1 First Come First Serve 19](#_Toc136014530)

[2.2.2 Round Robin (RR) 20](#_Toc136014531)

[2.2.3 Shortest Job First (SJF) 22](#_Toc136014532)

[2.2.4 Shortest Remain Time (SRT) 22](#_Toc136014533)

[2.3 Adaptive 23](#_Toc136014534)

[2.4 Định thời có xu hướng chuyển sang “Tweak and See” 25](#_Toc136014535)

[2.5 Định thời trên hệ thống Multiprocessor 26](#_Toc136014536)

[2.6 User Thread 28](#_Toc136014537)

[CHƯƠNG 3. CHƯƠNG TRÌNH MÔ PHỎNG HOẠT ĐỘNG CỦA CHIẾN LƯỢC FCFS 31](#_Toc136014538)

[3.1 Mục tiêu chương trình 31](#_Toc136014539)

[3.2 Xác định đầu ra, đầu vào của chương trình 31](#_Toc136014540)

[3.3 Tiến hành cài đặt và thực thi chương trình 31](#_Toc136014541)

[3.4 Các Test Case của chương trình 35](#_Toc136014542)

[3.5 Đánh giá và hướng phát triển 40](#_Toc136014543)

[3.5.1 Đánh giá 40](#_Toc136014544)

[3.5.2 Hướng phát triển 41](#_Toc136014545)

[TÀI LIỆU THAM KHẢO 42](#_Toc136014546)

# LỜI CÁM ƠN

Đầu tiên chúng em xin gửi lời cám ơn sâu sắc và chân thành tới các thầy cô trong trường Đại học Sư phạm, đặc biệt là thầy Trần Đức Tâm, một người thầy đã tận tâm, hết mình trực tiếp giảng dạy, hướng dẫn, động viên và giúp đỡ chúng em trong suốt quá trình thực hiện tiểu luận

Trong quá trình làm việc với Thầy, chúng em không những học hỏi những kiến thức mới bổ ích mà còn học được tinh thần và thái độ làm việc nghiêm túc trong một tập hợp nhóm.

Mặc dù chúng em đã cố gắng hoàn thiện đề tài này bằng tất cả sự nỗ lực và lượng kiến thức hiện có nên có thể sẽ không tránh khỏi những thiếu sót. Chúng em kính mong nhận được sự cảm thông và chỉ bảo của quý Thầy, cùng các bạn đồng môn.

TP Hồ Chí Minh, tháng 05 năm 2023

**Sinh Viên**

Phạm Hoài Nhân

Thái Gia Bảo

Mai Thành Danh

Trần Duy Quân

# DANH MỤC CÁC HÌNH VẼ

[Hình 1.1 Giao diện của Unix Version 7, 1979 chạy trong một trình giả lập 10](#_Toc136014467)

[Hình 1.2 Giao diện người dùng của Mac OS X 10.1, 2001 - phiên bản Mac OS đầu tiên xây dựng trên nền Unix 10](#_Toc136014468)

[Hình 1.3 Máy tính PDP-7, 1965 11](#_Toc136014469)

[Hình 1.4 Giản đồ về sử ra đời của Linux, nhưng không giới hạn đến các nhánh chính của Unix. Trong đó Linux không phải là "cháu ruột" (một nhánh chính thức) của Unix mà chỉ là "con nhận nuôi" (có thiết kế tương tự) mà thôi. 14](#_Toc136014470)

[Hình 1.5 Hệ điều hành Slackware được phát triển dựa trên mã nguồn Linux 15](#_Toc136014471)

[Hình 1.6 Biểu tượng chính thức của Linux 16](#_Toc136014472)

[Hình 3.1 Mô hình bài toán được cài đặt trong chương trình 30](#_Toc136014473)

[Hình 3.2 Giao diện đăng nhập 31](#_Toc136014474)

[Hình 3.3 Giao diện tuỳ chọn 31](#_Toc136014475)

[Hình 3.4 Giao diện nhập vào dữ liệu đối với trường hợp không có I/O 32](#_Toc136014476)

[Hình 3.5 Giao diện nhập vào dữ liệu đối với trường hợp có I/O 32](#_Toc136014477)

[Hình 3.6 Giao diện hiển thị kết quả đối với trường hợp không có I/O 33](#_Toc136014478)

[Hình 3.7 Giao diện hiển thị kết quả đối với trường hợp có I/O 34](#_Toc136014479)

# DANH MỤC CÁC BẢNG

[Bảng 1. Các Test Case ở tuỳ chọn không I/O 35](#_Toc136014480)

[Bảng 2. Các Test Case ở tuỳ chọn có I/O 37](#_Toc136014481)

# MỞ ĐẦU

1. **Tính cấp thiết của đề tài**

Hệ điều hành Linux là một hệ điều hành mã nguồn mở và miễn phí, phát triển dựa trên nhân Linux. Với nguồn gốc từ công việc của Linus Torvalds, Linux đã trở thành một trong những hệ điều hành phổ biến nhất trên thế giới. Với tính ổn định, bảo mật cao và khả năng tương thích với nhiều loại phần cứng, Linux đáp ứng được nhu cầu đa dạng của người dùng từ cá nhân đến doanh nghiệp lớn. Hệ điều hành này cung cấp một môi trường phát triển phong phú và hỗ trợ đa nền tảng, cho phép người dùng tuỳ chỉnh và tùy biến theo nhu cầu riêng. Với sự phát triển liên tục và sự đóng góp từ cộng đồng mã nguồn mở, Linux tiếp tục thúc đẩy sự tiến bộ và đóng vai trò quan trọng trong ngành công nghệ thông tin ngày nay.

Bài toán phân phối tài nguyên CPU cho các tiến trình là một trong những vấn đề quan trọng trong lĩnh vực hệ điều hành và quản lý tài nguyên hệ thống. Khi máy tính hoặc hệ thống có nhiều tiến trình cần thực thi, việc phân phối tài nguyên CPU một cách hiệu quả và công bằng giữa các tiến trình là rất quan trọng để đảm bảo hoạt động của hệ thống được ổn định và tối ưu. Bài toán này đặt ra một số vấn đề liên quan đến việc quyết định việc phân chia thời gian CPU giữa các tiến trình. Một trong những vấn đề quan trọng là việc lập lịch CPU, tức là quyết định xem tiến trình nào sẽ được thực thi trên CPU và trong khoảng thời gian nào. Mục tiêu là cung cấp sự công bằng và hiệu quả trong việc sử dụng tài nguyên CPU, đồng thời đảm bảo sự đáp ứng và độ ổn định của hệ thống.

Chính vì tầm quan trọng to lớn nên nhóm chúng em đã chọn đề tài: ***“Lịch sử ra đời, các giai đoạn phát triển của hệ điều hành Linux và các vấn đề liên quan đến bài toán phân phối tài nguyên CPU cho các tiến trình”.***

1. **Tình hình nghiên cứu**

Đối với việc tìm hiểu “Lịch sử ra đời, các giai đoạn phát triển của hệ điều hành Linux”, đã có nhiều nghiên cứu chi tiết về lịch sử và quá trình phát triển của Linux. Các tài liệu, sách và bài báo đã được xuất bản với mục đích phân tích sâu về các phiên bản Linux.

D. Bretthauer trong bài báo *“Open Source Software: A History”* [1]đã bàn về sự phát triển của ba hệ điều hành: GNU, BSD và Linux, cũng như cộng đồng đã phát triển xung quanh các hệ điều hành này và một số gói phần mềm phổ biến được phát triển bằng mô hình mã nguồn mở. Nó cũng thảo luận về một số nhân vật nổi tiếng trong lĩnh vực phần mềm mã nguồn mở và xác định cả "phần mềm miễn phí" và "phần mềm mã nguồn mở".

R. Hasan trong bài báo “History of Linux” [2] đã bàn chi tiết hơn về hệ điều hành Linux từ những ngày mà người sáng lập ra nó có những ý tưởng, những suy nghĩ như thế nào và dẫn tới quyết định tự mình tạo ra hệ điều hành Linux.

J. Aas [3] còn đi sâu hơn vào hệ thống, các mã lệnh, giải thuật và cách mà hệ điều hành Linux được thiết lập và hoạt động như thế nào một cách đầy đủ và chi tiết nhất, giải thích lí do tại sao mà Linux vẫn được ưa chuộng từ xưa cho đến nay mặc dù đã có nhiều hệ điều hành khác nổi lên.

Về các thuật toán lập lịch CPU được thiết kế và triển khai để quản lý tài nguyên CPU, đã có nhiều bài báo và các giáo trình nói về điều này rất chi tiết, tiểu biểu như các cuốn *“Nguyên lí Hệ điều hành”[4], “Giáo trình Hệ điều hành”[5], “Giáo trình lý thuyết hệ điều hành”[6], …* chứng tỏ vấn đề liên quan đến bài toán phân phối tài nguyên CPU cho các tiến trình đều đã được nghiên cứu rất đầy đủ ở trước đó.

1. **Đối tượng nghiên cứu và phạm vi nghiên cứu**
   1. **Đối tượng nghiên cứu**

* Hệ điều hành Linux.
* Các giải thuật phân phối tài nguyên CPU
* Chương trình mô phỏng chiến lược hoạt động của 1 chiến lược lập lịch CPU, cụ thể ở đây là chiến lược First Come First Serve (FCFS).
  1. **Phạm vi nghiên cứu**

Đề tài sẽ chỉ tập trung nghiên cứu lí thuyết về lịch sử ra đời, các giai đoạn phát triển của hệ điều hành Linux từ trước đến nay, và các bài toán, các giải thuật cụ thể để phân phối tài nguyên trong CPU.

Do giới hạn về thời gian và kiến thức, đề tài sẽ chỉ tập trung nghiên cứu, xây dựng chương trình minh hoạ một chiến lược hoạt động của chiến lược FCFS trong phân phối tài nguyên CPU.

1. **Phương thức nghiên cứu**
   1. **Phương pháp nghiên cứu lý thuyết:**

* Tìm hiểu các công trình nghiên cứu liên quan.
* Tìm hiểu chi tiết về lịch sử của hệ điều hành Linux
* Tìm hiểu về bài toán phân phối tài nguyên CPU

**4.2** **Phương pháp nghiên cứu thực nghiệm**:

* Tiến hành phân tích, xây dựng chương trình và cài đặt

1. **Nội dung tiểu luận**

Đề tài này gồm 3 chương:

**Chương 1:** Lịch sử ra đời và các giai đoạn phát triển của hệ điều hành Linux

**Chương 2:** Các vấn đề liên quan đến phân phối tài nguyên CPU cho các tiến trình

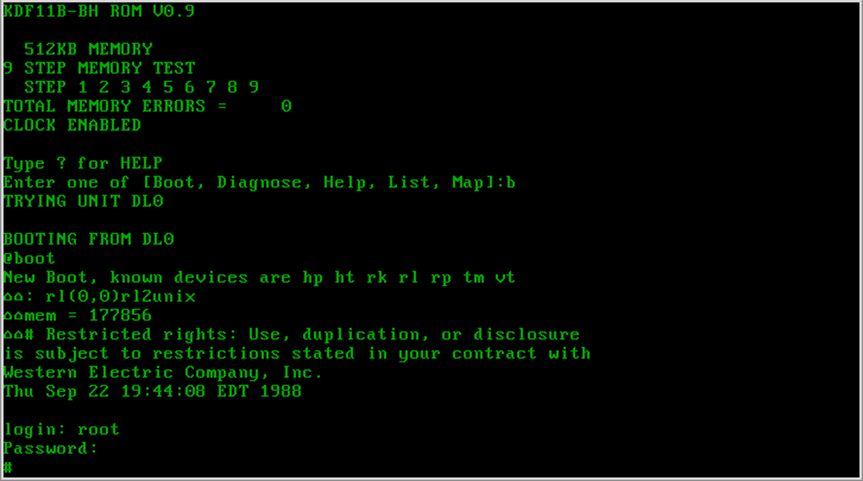
**Chương 3:** Chương trình mô phỏng hoạt động của chiến lược FCFS

# LỊCH SỬ RA ĐỜI VÀ CÁC GIAI ĐOẠN PHÁT TRIỂN CỦA HỆ ĐIỀU HÀNH LINUX

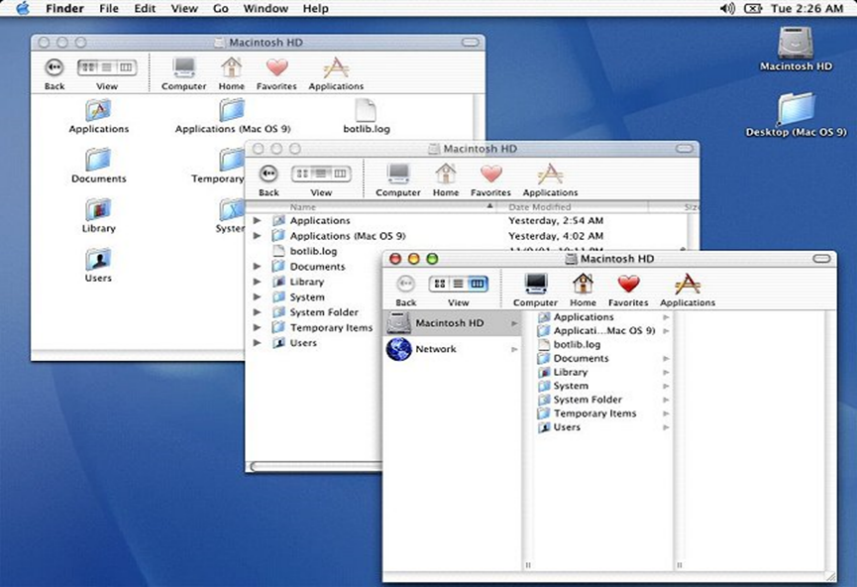
## Lịch sử ra đời của Linux

### Unix - tiền thân của Linux

Vào năm 1969, phòng thí nghiệm Bell Labs của AT&T (một trong những nhà cung cấp dịch vụ viễn thông lớn nhất Hoa Kỳ) đã phát triển một hệ điều hành gọi là "Unix" [7]. Unix ban đầu được viết bằng Assembly (Hợp ngữ), vốn khá khổ biến ở thời đó. Cần lưu ý MS-DOS được phát hành vào tận năm 1981 và 2 năm sau đó Windows mới ra đời. Đôi khi Unix được nhắc tới bởi ký tự 'X' (như trong A/UX hay Mac OS X).



Hình . Giao diện của Unix Version 7, 1979 chạy trong một trình giả lập



Hình . Giao diện người dùng của Mac OS X 10.1, 2001 - phiên bản Mac OS đầu tiên xây dựng trên nền Unix

Lúc đó, đĩa mềm vẫn còn là một thứ gì đó khá xa xỉ. Những chiếc máy tính thời đó đã "nhỏ gọn" hơn xưa, khi chỉ còn to bằng vài cái tủ quần áo ghép lại. Dữ liệu của chúng được nạp từ các băng từ (giống như các cuộn băng cát-xét khổng lồ), và thiết bị nhập/xuất chuẩn là các máy teletypewriter (nhìn giống như máy đánh chữ).



Hình . Máy tính PDP-7, 1965

Unix được thiết kế theo hướng module hóa "do one thing and do it well", gọi là "Unix Philosophy" (Triết lý Unix):

*1. Làm cho mỗi chương trình làm tốt một việc. Để thực hiện một công việc mới, hãy xây dựng lại từ đầu thay vì làm phức tạp các chương trình cũ bằng cách thêm các "tính năng" mới.*

*2. Mong đợi đầu ra của mọi chương trình trở thành đầu vào của một chương trình khác, chưa được biết đến. Đừng làm lộn xộn đầu ra với thông tin không liên quan. Tránh nghiêm ngặt các định dạng đầu vào dạng cột hoặc nhị phân. Đừng nhấn mạnh vào đầu vào tương tác.*

*3. Thiết kế và xây dựng phần mềm, thậm chí cả hệ điều hành, để dùng thử sớm, lý tưởng nhất là trong vòng vài tuần. Đừng ngần ngại vứt bỏ những phần vụng về và xây dựng lại chúng.*

*4. Ưu tiên sử dụng các công cụ hơn là trợ giúp không có kỹ năng để giảm bớt công việc lập trình, ngay cả khi bạn phải đi đường vòng để xây dựng các công cụ và mong muốn vứt bỏ một số trong số chúng sau khi sử dụng xong.*

Năm 1971, Unix đã được viết lại bằng ngôn ngữ C. Thực ra C ban đầu sinh ra để viết các phần mềm ứng dụng cho Unix.

Do Luật Chống độc quyền của chính phủ Mỹ, AT&T bị buộc phải công bố mã nguồn của Unix cho bất cứ người nào có nhu cầu. Kết quả là Unix đã phát triển nhanh chóng, được các tổ chức học thuật và doanh nghiệp sử dụng rộng rãi. Các nhánh lớn của Unix có thể kể đến như:

* Berkeley Software Distribution (BSD): được phát triển bởi Đại học California, Berkeley. BSD là một trong các nhánh lớn nhất của Unix còn sống tới hiện nay. BSD là nền tảng cốt lõi của các hệ điều hành Apple hiện đại như macOS, iOS, tvOS, …, firmware của các hệ máy console như PS4, Nintendo Switch, …
* Ngoài ra, chúng ta còn có Xenix (Microsoft), SunOS/Solaris (Sun Microsystems, hiện nay là Oracle), AIX (IBM), ...

Cần nói thêm một tí, các phần mềm hệ thống trong Unix có thể được đơn giản hóa (một cách rất đơn giản) thành 2 phần chính: kernel (hạt nhân) và bộ các công cụ hệ thống (system utilities, bao gồm các lệnh ls, cat, awk, find, grep, cd, shell, ...). Qua các phần mềm hệ thống (đang xét đến riêng nhóm này), người dùng có thể ra lệnh cho hạt nhân của hệ điều hành điều khiển phần cứng của máy tính để thực hiện các tác vụ tính toán cần thiết.

Năm 1984, lệnh chống độc quyền đối với AT&T đã kết thúc, nên họ đã biến Unix trở thành một phần mềm độc quyền (mã nguồn đóng) để kinh doanh.

Một năm trước đó (1983), một dự án phần mềm miễn phí là GNU Project (viết tắt đệ quy của GNU is Not Unix) đã được thiết lập bởi Richard Stallman nhằm mục đích viết ra một hệ điều hành tương thích hoàn toàn với Unix (gọi là Unix-like, hay "tương tự Unix"), toàn bộ được cấu thành bởi các phần mềm miễn phí (và mã nguồn mở).

Cần nhấn mạnh lại là GNU và các hệ điều hành Unix-like khác không phải là Unix mà chỉ được thiết kế dựa trên khung sườn của Unix. Nghĩa là, mục tiêu của dự án GNU là viết lại hoàn toàn một hệ điều hành có cấu trúc và hành vi "ăn theo" hệ điều hành Unix (bao gồm bộ công cụ cốt lõi như ls, cat, awk, find, grep, cd, ...) mà không kế thừa từ mã nguồn của Unix gốc như các hệ điều hành BSD, AIX, .... Do đó, GNU là một hệ điều hành POSIX-compliant.

Năm 1987, một giáo sư khoa học máy tính tên là Andrew S. Tanenbaum đã viết một hệ điều hành Unix-like tối giản tên là MINIX nhắm đến đối tượng các sinh viên muốn tìm hiểu về hệ điều hành. Tuy nhiên, nó bị giới hạn chỉ được sử dụng trong môi trường giáo dục.

Đầu những năm 90, đa phần các chương trình cơ bản của GNU Project như trình biên dịch, bộ thư viện, trình soạn thảo văn bản, command line shell, ... đã hoàn thiện. GNU chỉ còn thiếu một kernel tương thích với Unix để có thể tạo thành một hệ điều hành hoàn chỉnh. Kernel mà họ đang viết cho hệ điều hành này, gọi là GNU Hurd, đang lâm vào bế tắc.

Cần phải nói thêm, G++, trình biên dịch C++ được sử dụng phổ biến nhất hiện nay là một thành phần của GCC, viết tắt của GNU Compiler Collection (trước kia là GNU C Compiler).

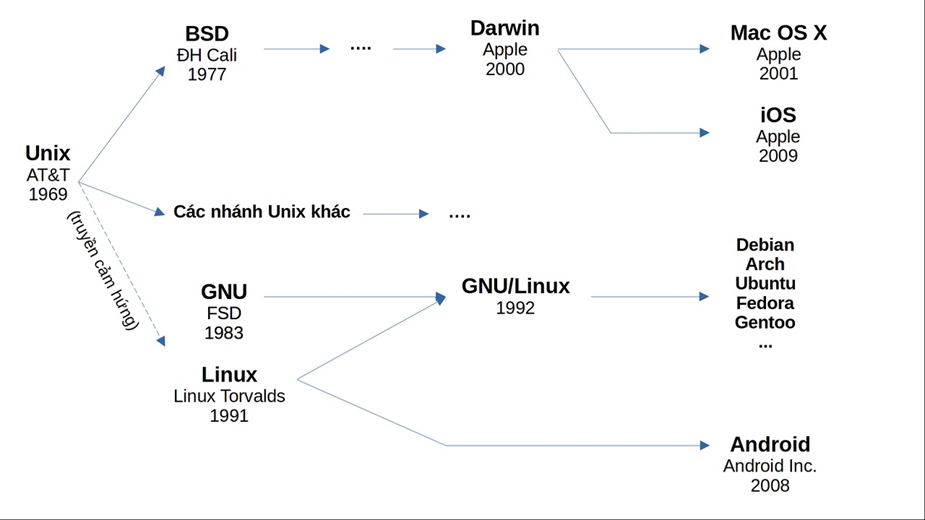
### Linux

Năm 1991, một chàng sinh viên trẻ đang học tại Đại học Helsinki (Phần Lan) tên là Linus Torvalds cảm thấy rất hứng thú với các thể loại hệ điều hành, đặc biệt là MINIX. Nhưng anh này cảm thấy khá bức xúc vì MINIX bị giới hạn chỉ được dùng trong môi trường giáo dục nên đã chơi lớn, viết ra luôn một kernel của riêng mình, và gọi nó là Linux [8].

Thời gian đầu, Linux sử dụng các phần mềm hệ thống từ dự án MINIX. Tuy nhiên khi đã có được nền tảng vững chắc, Linus, với sự trợ giúp của các lập trình viên, đã quyết định tích hợp các chương trình trong dự án GNU với kernel Linux của mình để tạo nên một hệ điều hành hoàn chỉnh, được phát hành với giấy phép GNU GPL, đảm bảo cho người dùng cuối tự do chạy, nghiên cứu, sửa đổi và chia sẻ các phần mềm.

Hệ điều hành hoàn chỉnh được cấu thành bởi kernel Linux và bộ phần mềm GNU được gọi là Hệ điều hành GNU/Linux, mà hiện nay thường được gọi tắt là Linux. Thực tế, Linux không phải là hệ điều hành mà chỉ là cái lõi bên trong mà thôi. Tất cả các "lệnh Linux" như cd, ls, cat, find, ... thực chất là các lệnh của bộ công cụ GNU. Nói chính xác hơn, đây là các lệnh của của tiêu chuẩn POSIX, được implement (triển khai) bởi dự án GNU. Các lệnh này sau đó mới giao tiếp với kernel (hạt nhân) của hệ điều hành, là Linux để ra lệnh cho phần cứng thực hiện các tác vụ cần thiết.

Cái lõi Linux đó có thể kết hợp với các bộ phần mềm khác để tạo ra các hệ điều hành khác nhau, ví dụ như Android được kết hợp từ kernel Linux và các thành phần khác do Google (ban đầu là Công ty Android) phát triển, không dùng các phần mềm từ bộ GNU. Có nghĩa là, Android cũng có các lệnh cd, ls, cat, ... nhưng các lệnh đó không phải là các lệnh của bộ công cụ GNU, mà là các lệnh của một bộ công cụ tên là toybox do Google phát triển. Toybox chỉ bao gồm một số lệnh cơ bản của tiêu chuẩn Unix nên rõ ràng không thể làm việc đầy đủ như một máy tính Linux thực thụ được, dù bản thân Android có thể được xem là một Linux distro (Linux Foundation và Google đều đồng ý về cách gọi này).



Hình . Giản đồ về sử ra đời của Linux, nhưng không giới hạn đến các nhánh chính của Unix. Trong đó Linux không phải là "cháu ruột" (một nhánh chính thức) của Unix mà chỉ là "con nhận nuôi" (có thiết kế tương tự) mà thôi.

## Các giai đoạn phát triển của hệ điều hành

Tháng 9/1991, phiên bản Linux 0.01, phiên bản Linux đầu tiên được Torvalds công bố. Tháng 10/1991 phiên bản Linux 0.02 ra đời.

Năm 1992, Torvals đã quyết định phát hành Linux dưới dạng mã nguồn mở của giấy phép GPL ( giấy phép phần mềm miễn phí ), cho phép tất cả mọi người có quyền download về để xem mã nguồn để cùng phát triển. Quyết định đã giúp hệ điều hành Linux phổ biến như ngày nay.

Năm 1993, Slackware là hệ điều hành đầu tiên phát triển dựa trên mã nguồn Linux ra đời. Slackware là một trong những hệ điều hành Linux đầu tiên và có tuổi đời lâu nhất hiện nay. Phiên bản mới nhất của Slackware được công bố vào tháng 5/2010.

Hình . Hệ điều hành Slackware được phát triển dựa trên mã nguồn Linux

Ngày 14/3/1994, sau 3 năm miệt mài, Torvalds cho ra mắt phiên bản hoàn thiện đầu tiên, Linux 1.0 với 176.250 dòng lệnh. Sau 1 năm, phiên bản 1.2 ra mắt với 310.950 dòng lệnh.

Ngày 3/11/1994, Red Hat Linux phiên bản 1.0 được giới thiệu. Đây là một trong những hệ điều hành được thương mại hóa đầu tiên dựa trên Linux.

Năm 1996, Linus Torvalds ghé thăm công viên hải dương học. Ông đã quyết định sử dụng hình ảnh chú chim cánh cụt để làm biểu tượng chính thức của Linux



Hình . Biểu tượng chính thức của Linux

Năm 1998, Linux bắt đầu được các tập đoàn công nghệ quan tâm và đầu tư để phát triển. Nổi bật là IBM đã đầu tư hàng tỉ USD để phát triển các dịch vụ và phần mềm trên nền tảng Linux. Ngoài IBM, còn có Compaq và Oracle cũng bắt đầu đầu tư và phát triển hê điều hành Linux.

Năm 2000-2005, Linus Torvalds được xuất hiện trên trang bìa của tạp chí về kinh tế BusinessWeek, với câu chuyện về sự thành công của hệ điều hành Linux. Giai đoạn này là thời kỳ Linux trở thành một trong những hệ điều hành phổ biến nhất trên thế giới. Nhiều doanh nghiệp và tổ chức chính phủ đã chuyển sang sử dụng Linux như một giải pháp cho hệ thống máy chủ của họ.

Năm 2006-2011: Là thời kỳ phát triển các tính năng mới cho Linux, như hệ thống file ext4, phát triển của Ubuntu và Fedora, sự ra đời của Android.

Năm 2012-2017: Phát triển của các công nghệ mới như OpenStack, Docker và các ứng dụng điện toán đám mây, điều này đã giúp Linux trở thành một trong những nền tảng quan trọng nhất cho các ứng dụng đám mây và máy chủ.

Năm 2018-nay: Giai đoạn hiện tại của Linux đang tập trung vào các công nghệ mới như học sâu, trí tuệ nhân tạo, IoT và các giải pháp an ninh. Linux vẫn tiếp tục là hệ điều hành phổ biến nhất trên thế giới và được sử dụng trong nhiều lĩnh vực khác nhau như máy chủ, đám mây, điện toán viễn thông và máy tính nhúng.

Tính đến hiện tại, Linux đã có rất nhiều phiên bản khác nhau, được xây dựng và phát triển riêng biệt bởi các công ty phần mềm và các cá nhân. Nổi bật trong số đó chính là hệ điều hành di động Android của Google, hiện là một trong những hệ điều hành thông dụng nhất hiện nay.

# CÁC VẤN ĐỀ LIÊN QUAN ĐẾN BÀI TOÁN PHÂN PHỐI TÀI NGUYÊN CPU CHO CÁC TIẾN TRÌNH

## Khái quát về bài toán

Bài toán phân phối tài nguyên CPU [9, 10] cho các tiến trình là một vấn đề quan trọng trong hệ thống máy tính. Khi có nhiều tiến trình chạy cùng lúc trên một máy tính, hệ thống phải phân chia tài nguyên CPU một cách hiệu quả để đảm bảo tính ổn định và tối ưu hóa hiệu suất của các tiến trình đang chạy. Sau đây là một số vấn đề liên quan đến bài toán phân phối tài nguyên CPU:

1. Lập lịch tiến trình: Hệ điều hành phải quyết định xem tiến trình nào được thực thi trước, tiến trình nào được thực thi sau đó. Các thuật toán lập lịch có thể được sử dụng để phân phối tài nguyên CPU cho các tiến trình. Một số thuật toán lập lịch phổ biến bao gồm Round Robin, Priority Scheduling, và Shortest Job First.
2. Thời gian chờ đợi: Thời gian chờ đợi của các tiến trình có thể ảnh hưởng đến hiệu suất của hệ thống. Nếu một tiến trình phải chờ đợi quá lâu để có được tài nguyên CPU, nó có thể làm giảm hiệu suất của hệ thống. Vì vậy, hệ thống phải có cơ chế phân phối tài nguyên CPU sao cho các tiến trình được thực thi một cách nhanh chóng và hiệu quả.
3. Tối ưu hiệu suất: Tối ưu hiệu suất của các tiến trình đang chạy là một yếu tố quan trọng trong bài toán phân phối tài nguyên CPU. Hệ thống phải sử dụng các thuật toán phân phối tài nguyên CPU sao cho các tiến trình có thể chạy với tốc độ cao nhất mà không làm giảm hiệu suất của hệ thống.
4. Đồng bộ hóa tiến trình: Đồng bộ hóa tiến trình là một yếu tố khác cần được xem xét trong bài toán phân phối tài nguyên CPU. Các tiến trình phải được đồng bộ hóa để tránh xung đột và đảm bảo tính ổn định của hệ thống.
5. Khối lượng tài nguyên CPU: Khối lượng tài nguyên CPU được yêu cầu bởi các tiến trình khác nhau có thể khác nhau. Các tiến trình yêu cầu nhiều tài nguyên CPU hơn có thể ảnh hưởng đến các tiến trình khác và làm giảm hiệu suất của hệ thống. Hệ thống cần phân phối tài nguyên CPU sao cho các tiến trình được cung cấp đủ tài nguyên để hoạt động hiệu quả.
6. Tối ưu hóa tiến trình đang chạy: Các tiến trình đang chạy có thể được tối ưu hóa để sử dụng tài nguyên CPU một cách hiệu quả hơn. Ví dụ, các tiến trình có thể được phân tích để tìm ra các phần của mã hoặc các thao tác được thực hiện một cách chậm chạp, từ đó cải thiện hiệu suất của chúng. Điều này có thể được thực hiện bằng cách sử dụng các công cụ phân tích và tối ưu hóa mã như Compiler Optimization hoặc Profilers.
7. Tài nguyên khác: Bên cạnh tài nguyên CPU, hệ thống cũng cần phân phối các tài nguyên khác như bộ nhớ, băng thông mạng và bộ xử lý đồ họa (GPU) để đảm bảo hiệu suất của các tiến trình. Sự cân bằng tài nguyên giữa các thành phần khác nhau của hệ thống cũng là một yếu tố quan trọng trong bài toán phân phối tài nguyên.
8. Tác động của ảo hóa: Khi sử dụng công nghệ ảo hóa, bài toán phân phối tài nguyên CPU trở nên phức tạp hơn. Việc phân phối tài nguyên CPU cho các máy ảo và các tiến trình chạy trong các máy ảo khác nhau cần được cân nhắc cẩn thận để đảm bảo tính ổn định và hiệu suất của hệ thống.

Trên đây là một số vấn đề liên quan đến bài toán phân phối tài nguyên CPU cho các tiến trình. Để giải quyết bài toán này, cần sử dụng các công cụ và thuật toán phân phối tài nguyên CPU phù hợp để đảm bảo tính ổn định và tối ưu hóa hiệu suất của hệ thống.

## Các thuật toán phân phối tài nguyên CPU

### First Come First Serve

Trong thuật toán này, độ ưu tiên phục vụ tiến trình căn cứ vào thời điểm hình thành tiến trình. Hàng đợi các tiến trình được tổ chức theo kiểu FCFS. Mọi tiến trình đều được phục vụ theo trình tự xuất hiện cho đến khi kết thúc hoặc bị ngắt. Ưu điểm của thuật toán này là giờ CPU không bị phân phối lại (không bị ngắt) và chi phí thực hiện thấp nhất (vì không phải thay đổi thứ tự ưu tiên phục vụ, thứ tự ưu tiên là thứ tự của tiến trình trong hàng đợi). Nhược điểm của thuật toán là thời gian trung bình chờ phục vụ của các tiến trình là như nhau (không kể tiến trình ngắn hay dài), do đó dẫn tới ba điểm sau:

* Thời gian chờ trung bình sẽ tăng vô hạn khi hệ thống tiếp cận tới hạn khả năng phục vụ của mình.
* Nếu độ phát tán thời gian thực hiện tiến trình tăng thì thời gian chờ đợi trung bình cũng tăng theo
* Khi có tiến trình dài, ít bị ngắt thì các tiến trình khác phải chờ đợi lâu hơn.

**Ví dụ:**

| **Tiến trình** | **Thời điểm vào** | **Thời gian xử lí** |
| --- | --- | --- |
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |

Thứ tự cấp phát tiến trình:

| **Tiến trình** | **P1** | **P2** | **P3** |
| --- | --- | --- | --- |
| Thời điểm | 0 | 24 | 27/30 |

Thời gian chờ trung bình: (23+25)/3=16.

### Round Robin (RR)

Giải thuật định thời luân phiên (Round - Robin Scheduling Algorithm - RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Tương tự như định thời FCFS nhưng sự trưng dụng CPU được thêm vào để chuyển CPU giữa các quá trình. Đơn vị thời gian nhỏ được gọi là định mức thời gian (time quantum) hay phần thời gian (time slice) được định nghĩa. Định mức thời gian thường từ 10 đến 100 mili giây. Hàng đợi sẳn sàng được xem như một hàng đợi vòng. Bộ định thời CPU di chuyển vòng quanh hàng đợi sẳn sàng, cấp phát CPU tới mỗi quá trình có khoảng thời gian tối đa bằng một định mức thời gian. Để cài đặt định thời RR, chúng ta quản lý hàng đợi sẳn sàng như một hàng đợi FCFS của các quá trình. Các quá trình mới được thêm vào đuôi hàng đợi. Bộ định thời CPU chọn quá trình đầu tiên từ hàng đợi sẳn sàng, đặt bộ đếm thời gian để ngắt sau 1 định mức thời gian và gởi tới quá trình. Sau đó, một trong hai trường hợp sẽ xảy ra. Quá trình có 1 chu kỳ CPU ít hơn 1 định mức thời gian. Trong trường hợp này, quá trình sẽ tự giải phóng. Sau đó, bộ định thời biểu sẽ xử lý quá trình tiếp theo trong hàng đợi sẳn sàng. Ngược lại, nếu chu kỳ CPU của quá trình đang chạy dài hơn 1 định mức thời gian thì độ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Chuyển đổi ngữ cảnh sẽ được thực thi và quá trình được đặt trở lại tại đuôi của hàng đợi sẳn sàng. Sau đó, bộ định thời biểu CPU sẽ chọn quá trình tiếp theo trong hàng đợi sẵn sàng.

**Ưu điểm:**

* Các quá trình sẽ được luân phiên cho CPU xữ lý nên thời gian chờ đợi sẽ ít.
* Đối với các quá trình liên quan đến nhập xuất, IO, người dùng thì rất hiệu quả.
* Việc cài đặt không quá phức tạp

**Nhược điểm:**

* Thời gian chờ đợi trung bình dưới chính sách RR thường là quá dài.
* Nếu thời gian định mức cho việc xữ lý quá lớn thì RR thành FIFO
* Nếu thời gian quá ngắn so với thời gian xữ lý của một tiến trình trong danh sách hàng đợi thì việc chờ đợi và xữ lý luân phiên sẽ nhiều.
* Qui tắc là định mức thời gian nên dài hơn 80% chu kỳ CPU.

**Ví dụ:**

| **Tiến trình** | **thời điểm vào** | **thời gian xử lý** |
| --- | --- | --- |
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |

Quantum = 4

Thì thứ tự cấp processor cho các tiến trình lần lượt là:

| **Tiến trình** | **P1** | **P2** | **P3** | **P1** | **P1** | **P1** | **P1** | **P1** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Thời điểm | 0 | 4 | 7 | 10 | 14 | 18 | 22 | 26 |

Vậy thời gian chờ đợi trung bình sẽ là: (3+5+6)/3 = 4,67 Như vậy RR có thời gian chờ đợi trung bình nhỏ hơn so với FCFS.

### Shortest Job First (SJF)

Một tiếp cận khác đối với việc định thời CPU là giải thuật định thời công việc ngắn nhất trước (shortest-job-first-SJF). Giải thuật này gán tới mỗi quá trình chiều dài của chu kỳ CPU tiếp theo cho quá trình sau đó. Khi CPU sẵn dùng, nó được gán tới quá trình có chu kỳ CPU kế tiếp ngắn nhất. Nếu hai quá trình có cùng chiều dài chu kỳ CPU kế tiếp, định thời FIFO được dùng. Chú ý rằng thuật ngữ phù hợp hơn là chu kỳ CPU kế tiếp ngắn nhất (shortest next CPU burst) vì định thời được thực hiện bằng cách xem xét chiều dài của chu kỳ CPU kế tiếp của quá trình hơn là toàn bộ chiều dài của nó. Chúng ta dùng thuật ngữ SJF vì hầu hết mọi người và mọi sách tham khảo tới nguyên lý của loại định thời biểu này như SJF.

**Ưu điểm:**

* Giải thuật được xem là tối ưu, thời gian chờ đợi trung bình giảm
* Tận dụng hết năng lực của CPU

**Nhược điểm:**

* Cài đặt thuật toán phức tạp, tốn nhiều xữ lý cho quá trình quản lý.
* Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp định thời CPU ngắn vì không có cách nào để biết chiều dài chu kỳ CPU tiếp theo.
* Giải thuật SJF có thể trưng dụng hoặc không trưng dụng CPU, dẫn tới giải thuật này có nhiều dị bản khác nhau và sẽ tối ưu hay không tối ưu phụ thuộc vào trưng dụng CPU.

### Shortest Remain Time (SRT)

Tương tự như SJF nhưng trong thuật toán này, độ ưu tiên thực hiện các tiến trình dựa vào thời gian cần thiết để thực hiện nốt tiến trình(bằng tổng thời gian trừ đi thời gian đã thực hiện). Như vậy, trong thuật toán này cần phải thường xuyên cập nhật thông tin về giời gian đã thực hiện của tiến trình. Đồng thời, chế độ phân bổ lại giờ CPU cũng phải được áp dụng nếu không sẽ làm mất tình ưu việc của thuật toán.

**Ưu điểm:**

* Thời gian chờ đợi, tồn tại trong hệ thống của mỗi tiến trình đều ngắn
* Thuật toán tối ưu nhất

**Nhược điểm:**

* Việc cài đặt thuật toán khá phức tạp
* Cần quản lý chặt chẽ việc điều phối các tiến trình
* Quản lý thời gian đến của mỗi tiến trình

## Adaptive

Trong hệ điều hành, khái niệm "adaptive" (có thể hiểu là "thích ứng") được sử dụng để chỉ các tính năng, thuật toán, hoặc cơ chế tự động điều chỉnh để thích nghi với môi trường hoặc điều kiện thay đổi [11, 12]. Adaptive có thể dự đoán thời gian sử dụng CPU thông qua quan sát quá khứ. Nếu trong quá khứ, tiến trình sử dụng CPU nhiều thì rất có thể trong tương lai sẽ sử dụng CPU nhiều, ngược lại nếu trong tiến trình I/O nhiều thì trong tương lai sẽ sử dụng CPU ít. Dự đoán thời gian tiến trình sử dụng CPU ở thời điểm thứ n được xác định dựa trên công thứ sau đây:

Trong bối cảnh của hệ điều hành, các tính năng adaptive có thể áp dụng trong nhiều khía cạnh, ví dụ:

* Adaptive Scheduling (Lập lịch thích ứng): Một thuật toán lập lịch thích ứng có khả năng điều chỉnh thời gian phân chia CPU cho các tiến trình dựa trên tình trạng và yêu cầu của hệ thống. Thuật toán này có thể thay đổi ưu tiên của các tiến trình, thời gian được cấp phát cho mỗi tiến trình, hoặc áp dụng các chiến lược linh hoạt khác để tối ưu hiệu suất hoặc đáp ứng của hệ thống.
* Adaptive Power Management (Quản lý năng lượng thích ứng): Hệ điều hành có thể sử dụng các cơ chế quản lý năng lượng thích ứng để điều chỉnh hoạt động của các thành phần phần cứng như CPU, màn hình, hay ổ cứng. Các cơ chế này có thể điều chỉnh tần số hoặc hiệu suất hoạt động của các thành phần dựa trên tình trạng sử dụng và yêu cầu của hệ thống, từ đó giảm tiêu thụ năng lượng và tăng tuổi thọ pin.
* Adaptive Memory Management (Quản lý bộ nhớ thích ứng): Quản lý bộ nhớ thích ứng đề cập đến việc sử dụng các thuật toán và cơ chế linh hoạt để điều chỉnh việc phân bổ bộ nhớ cho các tiến trình và ứng dụng. Hệ điều hành có thể tự động điều chỉnh việc cấp phát, giải phóng và tái sử dụng bộ nhớ dựa trên yêu cầu của các tiến trình và tình trạng sử dụng bộ nhớ.

Các tính năng adaptive trong hệ điều hành nhằm cung cấp hiệu suất tối ưu, tương thích linh hoạt và sử dụng tài nguyên hiệu quả trong các môi trường thay đổi và đa dạng.

Dưới đây là một số ví dụ về tính năng adaptive trong hệ điều hành:

* Adaptive Display Brightness (Độ sáng màn hình thích ứng): Một tính năng phổ biến trên các thiết bị di động là điều chỉnh độ sáng của màn hình dựa trên môi trường xung quanh. Hệ điều hành sử dụng các cảm biến ánh sáng để đo mức độ sáng xung quanh và điều chỉnh độ sáng màn hình để tiết kiệm năng lượng và cung cấp trải nghiệm người dùng tốt hơn.
* Adaptive Battery Management (Quản lý pin thích ứng): Trên các thiết bị di động, hệ điều hành có thể áp dụng các chiến lược quản lý pin thích ứng để kéo dài thời gian sử dụng pin. Ví dụ, hệ điều hành có thể tắt các ứng dụng chạy ngầm, giảm tần số hoạt động của CPU hoặc hạn chế các hoạt động không cần thiết để giảm tiêu thụ năng lượng.
* Adaptive Scheduling Algorithms (Thuật toán lập lịch thích ứng): Một ví dụ về thuật toán lập lịch thích ứng là Completely Fair Scheduler (CFS) trong hệ điều hành Linux. CFS sử dụng một cây đỏ đen (red-black tree) để theo dõi thời gian sử dụng CPU của các tiến trình và điều chỉnh ưu tiên của chúng. Khi một tiến trình cần thực thi thời gian CPU nhiều hơn, CFS tăng độ ưu tiên của nó để đảm bảo công bằng và tối ưu hiệu suất.
* Adaptive Memory Paging (Phân trang bộ nhớ thích ứng): Trong hệ điều hành, việc quản lý trang bộ nhớ (paging) có thể thích ứng với hoạt động và yêu cầu của các tiến trình. Hệ điều hành có thể điều chỉnh kích thước bộ nhớ trang (page size), tần suất truy cập vào các trang (page access frequency), hoặc cơ chế thay thế trang (page replacement algorithm) để tối ưu hiệu suất và sử dụng bộ nhớ.

Các ví dụ trên chỉ ra cách mà hệ điều hành có thể tự động điều chỉnh và thích ứng với môi trường, tình trạng hoạt động và yêu cầu của các thành phần khác nhau để cung cấp hiệu suất tốt nhất và tối ưu hóa sử dụng tài nguyên.

## Định thời có xu hướng chuyển sang “Tweak and See”

Vấn đề định thời trong hệ điều hành liên quan đến việc quyết định thứ tự thực hiện các tiến trình và phân chia tài nguyên CPU cho chúng. Theo truyền thống, các thuật toán định thời đã được phát triển để đảm bảo công bằng, hiệu suất và đáp ứng trong việc phân phối tài nguyên. Tuy nhiên, gần đây, một xu hướng đang chuyển sang phương pháp “Tweak and See” trong việc điều chỉnh các thông số định thời.

Phương pháp “Tweak and See” ám chỉ việc thay đổi và thử nghiệm các thông số định thời để tìm ra cấu hình tối ưu. Thay vì dựa trên các thuật toán định thời cố định, phương pháp này tạo ra một không gian tham số lớn để điều chỉnh các thông số như độ ưu tiên của tiến trình, thời gian cấp phát CPU, hoặc ngưỡng thời gian chờ.

Việc sử dụng phương pháp “Tweak and See” cho phép hệ điều hành tìm ra cấu hình tối ưu cho một môi trường cụ thể. Thông qua việc thực hiện các thay đổi và đánh giá hiệu suất, hệ điều hành có thể điều chỉnh thông số định thời để đáp ứng tốt nhất với yêu cầu và tình trạng của hệ thống.

Tuy nhiên, phương pháp này cũng đòi hỏi quá trình thử nghiệm và điều chỉnh liên tục để tìm ra cấu hình tối ưu và có thể tốn nhiều thời gian và tài nguyên. Do đó, việc sử dụng phương pháp “Tweak and See” trong vấn đề định thời cần cân nhắc kỹ lưỡng và đòi hỏi kiến thức sâu về hệ thống và môi trường để đảm bảo rằng các thay đổi được thực hiện một cách an toàn và có hiệu quả. Nếu điều chỉnh sai hoặc quá mức có thể gây ra sự bất ổn, hoặc quá nhiều vấn đề khác.

Dưới đây là một ví dụ để minh họa phương pháp “Tweak and See” trong vấn đề định thời:

Giả sử bạn đang làm việc trên một hệ thống máy chủ và bạn muốn điều chỉnh các thông số định thời để cân bằng công việc giữa các tiến trình. Thay vì sử dụng một thuật toán định thời cố định, bạn quyết định áp dụng phương pháp “Tweak and See” để điều chỉnh các thông số định thời.

* Bước 1: Bạn bắt đầu bằng việc chọn một thông số định thời cụ thể để điều chỉnh, ví dụ: độ ưu tiên của các tiến trình.
* Bước 2: Thiết lập giá trị ban đầu cho thông số định thời và theo dõi hiệu suất hệ thống.
* Bước 3: Tiếp theo, thay đổi giá trị thông số định thời và quan sát sự thay đổi trong hiệu suất và cân bằng công việc. Bạn có thể tăng hoặc giảm độ ưu tiên của các tiến trình để xem sự ảnh hưởng của các thay đổi này.
* Bước 4: Tiếp tục điều chỉnh các thông số định thời theo cách tuần tự và quan sát hiệu suất hệ thống sau mỗi thay đổi.
* Bước 5: Dựa trên quan sát và đánh giá hiệu suất, bạn lựa chọn giá trị tốt nhất cho thông số định thời, tức là cấu hình tối ưu cho việc cân bằng công việc giữa các tiến trình.
* Bước 6: Bạn tiếp tục với các thông số định thời khác và lặp lại quá trình điều chỉnh và đánh giá cho đến khi bạn tìm ra các giá trị tối ưu cho tất cả các thông số định thời.

Với phương pháp “Tweak and See”, bạn có thể tùy chỉnh các thông số định thời khác nhau, như thời gian cấp phát CPU, ngưỡng thời gian chờ, hoặc quy tắc lập lịch, để tìm ra cấu hình tốt nhất cho việc quản lý tiến trình và tối ưu hiệu suất hệ thống.

## Định thời trên hệ thống Multiprocessor

Định thời trên hệ thống Multiprocessor liên quan đến việc quản lý và phân chia tài nguyên CPU cho các tiến trình trên một hệ thống có nhiều bộ xử lý (processor).

Trong hệ thống Multiprocessor, có nhiều lõi CPU độc lập hoạt động đồng thời, cho phép thực thi đa nhiệm song song và tăng hiệu suất hệ thống. Tuy nhiên, để đạt được hiệu suất tối đa, cần có một cơ chế định thời hiệu quả để phân phối công việc cho các lõi CPU một cách cân bằng.

Có hai phương pháp phổ biến để định thời trên hệ thống Multiprocessor:

* Định thời tĩnh (Static Scheduling): Trong phương pháp này, công việc được chia thành các phần bằng nhau và gán cho các lõi CPU theo một lịch trình cố định. Các lõi CPU được sử dụng với tần suất cố định và không thay đổi trong quá trình thực thi. Định thời tĩnh đơn giản và dễ triển khai, nhưng không linh hoạt trong việc ứng phó với biến đổi tải công việc và yêu cầu thời gian thực.
* Định thời động (Dynamic Scheduling): Trái ngược với định thời tĩnh, định thời động cho phép điều chỉnh phân phối công việc và tài nguyên CPU dựa trên tình trạng và yêu cầu thời gian thực của hệ thống. Các thuật toán định thời động như lập lịch thích ứng (Adaptive Scheduling) và di chuyển luồng (Thread Migration) được sử dụng để cân bằng công việc giữa các lõi CPU, tối ưu hóa hiệu suất và đáp ứng của hệ thống.

Trong định thời động, các quyết định phân phối công việc và tài nguyên CPU có thể được điều chỉnh trong thời gian thực, dựa trên mức độ ưu tiên của tiến trình, tình trạng tải công việc của các lõi CPU và yêu cầu thời gian thực của các tiến trình. Việc điều chỉnh độ ưu tiên, di chuyển luồng và cân bằng công việc đảm bảo rằng tài nguyên CPU được sử dụng một cách hiệu quả và đáp ứng yêu cầu của hệ thống.

Định thời trên hệ thống Multiprocessor là một lĩnh vực phức tạp và có nhiều thuật toán và chiến lược khác nhau được phát triển để đạt được hiệu suất tối ưu. Sự lựa chọn của phương pháp và thuật toán định thời phụ thuộc vào yêu cầu cụ thể của ứng dụng và hệ thống.

Một ví dụ về bài toán định thời trên hệ thống Multiprocessor là bài toán lập lịch đa nhiệm trên môi trường đa lõi (Multicore Environment). Giả sử bạn có một hệ thống máy tính với ba lõi CPU và một danh sách các tiến trình cần thực thi. Mỗi tiến trình có một thời gian thực thi khác nhau và yêu cầu một số lõi CPU để hoàn thành công việc.

Bài toán là phân chia các tiến trình vào các lõi CPU sao cho tối đa hóa hiệu suất và đáp ứng của hệ thống. Đồng thời, cần đảm bảo rằng không có lõi CPU nào bị bỏ trống hoặc quá tải.

Ví dụ:

Có ba tiến trình A, B và C cần thực thi trên hệ thống có ba lõi CPU.

Tiến trình A cần 2 đơn vị thời gian để hoàn thành.

Tiến trình B cần 3 đơn vị thời gian để hoàn thành.

Tiến trình C cần 4 đơn vị thời gian để hoàn thành.

Mỗi lõi CPU chỉ có thể thực thi một tiến trình trong mỗi đơn vị thời gian.

Bạn cần phân chia các tiến trình vào các lõi CPU sao cho thời gian hoàn thành của hệ thống là nhỏ nhất.

Một lịch trình lập lịch tối ưu có thể là:

Lõi CPU 1: Tiến trình A

Lõi CPU 2: Tiến trình B

Lõi CPU 3: Tiến trình C

Với lịch trình này, thời gian hoàn thành của hệ thống là 4 đơn vị thời gian (tổng thời gian lâu nhất để hoàn thành một tiến trình). Các lõi CPU được sử dụng hiệu quả và không có lõi CPU nào bị bỏ trống hoặc quá tải.

## User Thread

User Thread (tiến trình người dùng) [13, 14] là một dạng tiến trình được quản lý và thực thi bởi ứng dụng người dùng mà không cần sự can thiệp từ hạt nhân hệ điều hành. User thread đóng vai trò quan trọng trong việc phân chia thời gian CPU giữa các tiến trình và quyết định thời gian thực thi chúng. Tuy nhiên, User Thread cũng có một số vấn đề cần được xem xét trong quản lý tiến trình:

Việc quản lý User Thread vẫn có tác động đến lập lịch CPU. Khi một user thread được chọn để thực thi, hệ điều hành sẽ cấp phát thời gian CPU cho tiểu trình đó. Nếu một User Thread bị chặn (blocked), hệ điều hành có thể chuyển sang thực thi các tiểu trình khác. Khi User Thread giải chặn (unblock), hệ điều hành có thể quyết định thực thi tiếp User Thread đó hoặc tiếp tục với các tiểu trình khác.

Đồng bộ hóa và xử lý tài nguyên: Vì User Thread không phải là các tiến trình hạt nhân, nên chúng không thể sử dụng trực tiếp các tài nguyên như CPU, bộ nhớ, I/O. Thay vào đó, User Thread phải chia sẻ tài nguyên với tiến trình người dùng chủ. Điều này có thể dẫn đến các vấn đề về đồng bộ hóa và quản lý tài nguyên, bởi vì các User Thread phải tự đảm bảo rằng việc truy cập vào tài nguyên chia sẻ được thực hiện một cách an toàn và không xảy ra xung đột.

Hiệu suất và đáp ứng: Trong trường User Thread được triển khai bằng cách sử dụng thư viện đa luồng trong ứng dụng người dùng, một vấn đề có thể xảy ra là khi một User Thread chờ đợi hoặc bị chặn, tất cả các User Thread khác cũng bị chặn. Điều này có thể ảnh hưởng đến hiệu suất và đáp ứng của ứng dụng, vì các user thread không thể tiếp tục thực thi trong khi một user thread khác đang chờ đợi hoặc bị chặn.

Quản lý lập lịch: Trong quản lý tiến trình, lập lịch User Thread cũng có thể trở thành một vấn đề. Vì các User Thread được quản lý bởi ứng dụng người dùng, hệ điều hành không có thông tin chi tiết về User Thread và không thể can thiệp để lập lịch thực thi chúng trên CPU. Điều này có thể dẫn đến việc không có sự cân đối trong việc sử dụng CPU giữa các User Thread, và một User Thread có thể chiếm quá nhiều thời gian CPU trong khi các User Thread khác phải chờ đợi lâu hơn.

Quản lý lỗi và gỡ lỗi: Khi xảy ra lỗi trong một User Thread, việc quản lý và gỡ lỗi có thể trở nên phức tạp hơn. Hệ điều hành không có kiến thức về User Thread và không thể kiểm soát và theo dõi trạng thái của chúng. Điều này có thể làm cho việc gỡ lỗi và xác định nguyên nhân gốc rễ của lỗi trở nên khó khăn hơn.

Để giải quyết các vấn đề liên quan đến User Thread trong quản lý tiến trình, có thể áp dụng các biện pháp sau:

Sử dụng hỗ trợ từ hệ điều hành: Một số hệ điều hành cung cấp hỗ trợ cho việc quản lý User Thread thông qua các API hoặc thư viện đa luồng. Sử dụng các API này có thể giúp đồng bộ hóa và quản lý tài nguyên tốt hơn trong quá trình thực thi User Thread.

Lập lịch và ưu tiên User Thread: Ứng dụng có thể tự định nghĩa cách lập lịch và ưu tiên User Thread bên trong nó. Bằng cách thiết kế một thuật toán lập lịch phù hợp, việc phân chia thời gian CPU giữa các User Thread có thể được cân bằng hơn, đảm bảo rằng không có User Thread nào chiếm quá nhiều thời gian CPU và đáp ứng tốt hơn.

Sử dụng các kỹ thuật đồng bộ hóa: Để đảm bảo đồng bộ và tránh xung đột tài nguyên giữa các User Thread, có thể sử dụng các kỹ thuật đồng bộ hóa như khóa (lock), biến điều kiện (condition variable), semaphore, hoặc mutex để quản lý truy cập vào tài nguyên chia sẻ. Điều này giúp đảm bảo rằng User Thread không gây ra xung đột và tránh thời gian chờ đợi không cần thiết.

Gỡ lỗi và quản lý lỗi: Để giải quyết vấn đề gỡ lỗi và quản lý lỗi trong User Thread, có thể sử dụng các công cụ và kỹ thuật phù hợp. Ví dụ, việc ghi log, đánh dấu user thread và theo dõi trạng thái của chúng, sử dụng công cụ gỡ lỗi phù hợp có thể giúp xác định và sửa lỗi một cách hiệu quả.

Tối ưu hóa hiệu suất: Để đảm bảo hiệu suất tốt của các User Thread, có thể áp dụng các phương pháp tối ưu hóa như phân tích và cải tiến thuật toán, sử dụng kỹ thuật đa luồng hiệu quả, hoặc tận dụng các tài nguyên phần cứng một cách tối đa.

# CHƯƠNG TRÌNH MÔ PHỎNG HOẠT ĐỘNG CỦA CHIẾN LƯỢC FCFS

## Mục tiêu chương trình

Mục tiêu của chương trình là mô phỏng lại được hoạt động của chiến lược FCFS dựa trên các tiến trình cho trước.

Về phần mềm, sử dụng ngôn ngữ C++ viết trên chương trình Microsoft Visual Studio Code 2022.

## Xác định đầu ra, đầu vào của chương trình

Input của chương trình là thuật toán các có trường hợp I/O hay ko, nếu không sẽ tiến hành nhập các tiến trình bao gồm các thông tin như tên, thời điểm vào Ready List (RL) và thời gian CPU , nếu có thì sẽ thêm vào thời điểm I/O và thời gian I/O của tiến trình.

Output của chương trình sẽ là sơ đồ Grant thể hiện việc sắp xếp các tiến trình CPU, các thông số về thời gian chờ trung bình, thời gian phản hồi trung bình và thời gian xoay vòng trung bình.

A picture containing text, line, diagram, screenshot

Description automatically generated

Hình . Mô hình bài toán được cài đặt trong chương trình

## Tiến hành cài đặt và thực thi chương trình

Áp dụng lý thuyết đã được nói ở phần 2.2.1, chương trình được cài đặt bao gồm phần thuật toán và giao diện sau.

Khi chạy chương trình, chương trình sẽ yêu cầu chúng ta nhập dữ kiện có I/O hay không, rồi mới nhập nhập các dữ liệu bao gồm: số các tiến trình, thông tin về Arrival Time, Burst Time và thời điểm I/O, thời gian I/O (đối với lựa chọn có I/O) của từng các tiến trình ấy.

A screenshot of a computer

Description automatically generated with medium confidence

Hình . Giao diện đăng nhập

A screen shot of a computer

Description automatically generated with low confidence

Hình . Giao diện tuỳ chọn

A screen shot of a computer

Description automatically generated with low confidence

Hình . Giao diện nhập vào dữ liệu đối với trường hợp không có I/O

A screen shot of a computer program

Description automatically generated with low confidence

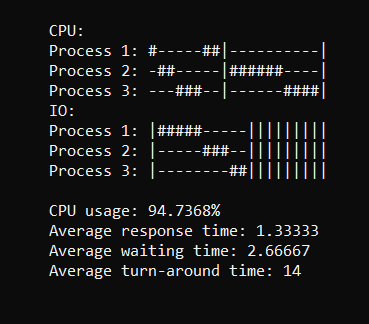
Hình . Giao diện nhập vào dữ liệu đối với trường hợp có I/O

Sau khi nhập dữ liệu xong, chương trình sẽ thực thi và bắt đầu tiến hành giải thuật FCFS, sau đó sẽ hiện ra bảng kết quả các thông tin bao gồm: sơ đồ Grant thể hiện việc sắp xếp các tiến trình ở Ready List và I/O List (đối với trường hợp có I/O), CPU usage (Hiệu suất của CPU, Average Waitng Time (Thời gian chờ trung bình), Average Response Time (Thời gian phản hồi trung bình), Average Turn - Around Time (Thời gian xoay vòng trung bình).

A screen shot of a computer

Description automatically generated with low confidence

Hình . Giao diện hiển thị kết quả đối với trường hợp không có I/O



Hình . Giao diện hiển thị kết quả đối với trường hợp có I/O

## Các Test Case của chương trình

Bảng sau đây thể hiện một số các Test Case ở 2 tuỳ chọn của chương trình:

Bảng . Các Test Case ở tuỳ chọn không I/O

|  |  |  |
| --- | --- | --- |
| **TEST CASE** | **INPUT** | **OUTPUT** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

Bảng . Các Test Case ở tuỳ chọn có I/O

|  |  |  |
| --- | --- | --- |
| **TEST CASE** | **INPUT** | **OUTPUT** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

## Đánh giá và hướng phát triển

### Đánh giá

Chương trình hoạt động tốt với hầu như mọi Test Case, xử lí các thông số đầu vào mà không gặp trục trặc hay vấn đề gì cả, trừ tuỳ chon I/O có khá nhiều Test Case bị sai do việc lập trình thuật toán chưa tốt.

Chương trình đơn giản, dễ dùng, chỉ việc nhập số liệu, xử lí và hiện kết quả

Tuy nhiên do vấn đề hạn chế về thời gian nên đã không có giao diện tốt hơn cho chương trình, giúp cho chương trình mô phỏng có giao diện ưa nhìn hơn, dễ sử dụng về sau này hơn..

Vì quy mô của chương trình chỉ ở mức mô phỏng lại thuật toán nên không có cơ sở dữ liệu để lưu trữ các bài toán để sau này có thể tìm kiếm lại được, giúp việc tìm ra kết quả nhanh chống hơn

### Hướng phát triển

Khắc phụ những hạn chế nêu trên, cụ thể như tạo thêm giao diện giúp cho người dùng có thể dễ dàng sử dụng hơn.

Thêm vào chương trình một số thuật toán khác như SJT, RR, tối ưu thuật toán FCFS có I/O hơn giúp cho chương trình có thể giải quyết ở mức độ nhiều bài toán, giải thuật khác nhau.

# TÀI LIỆU THAM KHẢO

[1] D. Bretthauer, "Open source software: A history," 2001.

[2] R. Hasan, "History of linux," ed, 2002.

[3] J. J. R. O. Aas, "Understanding the Linux 2.6. 8.1 CPU scheduler," vol. 16, pp. 1-38, 2005.

[4] H. Đ. Phương, *"Nguyên lý Hệ điều hành"*. NXB Giáo dục, 2010.

[5] T. M. Phương, *"Giáo trình Hệ điều hành"*. NXB Học viện Công nghệ Bưu chính Viễn thông, 2013.

[6] N. K. Tuấn, *"Giáo trình lý thuyết Hệ điều hành"*. NXB ĐHQGHN, 2014.

[7] Hiraki. (2021, May 18). *[Series] Sự kỳ diệu của Linux (Phần 1: Lịch sử và tính ưu việt của Linux)*. Available: <https://spiderum.com/bai-dang/Series-Su-ky-dieu-cua-Linux-Phan-1-uju>

[8] Đ. Nguyễn. (2018, May 18). *Lịch sử về hệ điều hành Linux*. Available: <https://blogd.net/linux/lich-su-he-dieu-hanh-linux/>

[9] L. Đ. Tiến. (2017, May 18). *Tiến trình trong hệ điều hành (Phần 3)*. Available: <https://viblo.asia/p/tien-trinh-trong-he-dieu-hanh-phan-3-3Q75wg6Q5Wb>

[10] L. K. N. Ân, *"Hệ điều hành cơ bản"*. ĐHKHTN TPHCM.

[11] D. K. Yau and S. S. Lam, "Adaptive rate-controlled scheduling for multimedia applications," in *Proceedings of the fourth ACM international conference on multimedia*, 1997, pp. 129-140.

[12] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th international conference on Autonomic computing*, 2009, pp. 117-126.

[13] Q. Anh. (2021, May 18). *Core và Thread là gì? Ảnh hưởng thế nào đến sức mạnh CPU?* Available: <https://fptshop.com.vn/tin-tuc/danh-gia/core-va-thread-la-gi-139997>

[14] Đ. Tài. (2022, May 18). *Giới thiệu về thread*. Available: <https://viblo.asia/p/gioi-thieu-ve-thread-aNj4vXpdL6r>