TRƯỜNG ĐẠI HỌC THỦY LỢI

Khoa Công Nghệ Thông Tin (CSE106)



Bộ môn: Nguyên lý hệ điều hành

Giảng Viên: Vũ Thành Vinh

Chủ đề: Tiến trình và Luồng

Nhóm thực hiện: Nhóm 4

Lớp tín chỉ : 63TH3

Các thành viên

Hoàng Ánh Nguyệt 2151062839

Nguyễn Thị Ngọc Ánh 2151062717

Nguyễn Thị Phương 2151062850

Cao Thị Khánh Linh 2151060182

Lê Khắc Ninh 2151062844

**Hà Nội, Tháng 12 năm 2023**

**MỤC LỤC**

1 – Tiến trình

Một tiến trình về cơ bản là một chương trình đang được thực thi. Việc thực hiện một tiến trình phải diễn ra theo kiểu tuần tự.Nói một cách đơn giản, chúng ta viết các chương trình máy tính của mình vào một tệp văn bản và khi chúng ta thực thi chương trình này, nó sẽ trở thành một quá trình thực hiện tất cả các tác vụ được đề cập trong chương trình. Khi một chương trình được tải vào bộ nhớ và trở thành một tiến trình, nó có thể được chia thành bốn phần ─ Stack, Heap, Text và dữ Data.

Stack: Ngăn xếp tiến trình chứa dữ liệu tạm thời như tham số phương thức, hàm địa chỉ trả về và các biến cục bộ

Heap: Đây là bộ nhớ được cấp phát động cho một tiến trinh trong thời gian chạy của nó

Text: Bao gồm các hoạt động hiện tại được biểu thị bằng giá trị của bộ đếm chương trình và nội dung của các thanh ghi của bộ xử lý

Data: Chứa các biến toàn cục và tĩnh

Hãy lấy một VD về một chương trình đơn giảm được viết bằng Python

A screen shot of a computer

Description automatically generated

Chúng ta có thể thấy ở đây biến name là một biến toàn cục nên nó sẽ được lưu ở trong Data

Tiếp theo chúng ta có một hàm “sayHello” với paramenter là name nó sẽ được lưu trong Stack

Sau khi chúng ta gọi lại hàm “sayHello” để thực thi trong thời gian nó thực thi thì nó sẽ được lưu trong Heap

* 1. Mô hình tiến trình

Các mô hình tiến trình gồm

* Thực hiện tuần tự: Tại một thời điểm chỉ có thể thực hiệnmột chương trình, khi chương trình đóthực hiện xong thì mới chạy được chương trình khác (Chế độ mặc định của hầu hết các ngôn ngữ lập trình)
* Thực hiện song song: Có thể chạy nhiều chương trình cùng một lúc (hệ thống đa xử lý)
  1. Thông tin về một tiến trình

Thông tin về một tiến trình gồm:

* + - Số định danh của tiến trình (PID-Process Identification) cho phép phân biệt với tiến trình khác
    - Trạng thái tiến trình: một trong năm trạng thái liệt kê ở phần trước
    - Nội dung một số thanh ghi CPU: thanh ghi con trỏ lệnh,con trỏ ngăn xếp, thanh ghi dữ liệu, thanh ghi trạng thái
    - Thông tin về bộ nhớ của tiến trình: chỉ ra nơi chứa tiến trình
    - Thông tin phục vụ việc điều độ tiến trình: mức độ ưu tiên của tiến trình, vị trí tiến trình trong các hàng đợi, tài nguyên tiến trình đang sở hữu
    - Danh sách các tài nguyên khác: bao gồm danh sách các file đang mở của tiến trình, các thiết bị vào ra tiến trình đang sử dụng
    - Thông tin thống kê phục vụ quản lý: thông tin loại này thường được sử dụng phục vụ thống kê hoặc tính toán chi phí đối với các hệ thống dùng chung (như khi đi thuê máy tính) và bao gồm thông tin về thời gian sử dụng CPU, giới hạn thời gian, tài khoản của người sở hữu tiến trình .v.v

1.3 - Vòng đời của một tiến trình

Khi một tiến trình thực thi, nó sẽ chuyển qua các trạng thái khác nhau. Các giai đoạn này có thể khác nhau ở các hệ điều hành khác nhau và tên của các trạng thái này cũng không được chuẩn hóa.

Mỗi tiến trình sẽ bao gồm 5 trạng thái sau

Start: Đây là trạng thái ban đầu khi một quá trình được bắt đầu (khởi tạo lần đầu)

Ready: Các tiên trình đã sẳn sàng đang chờ hệ điều hành cấp phát bộ xử lý để chúng có thể chạy. Quá trình có thể rơi vào trạng thái này sau trạng thái Start hoặc trong khi đang chạy nhưng bị gian đoạn bởi bộ lập lịch để gán CPU cho một tiến trình khác

Running: Khi tiến trình đã được lập lịch hệ điều hành sẽ gán cho bộ xử lý trạng thái quy trình được đặt thành đang chạy và bộ xử lý sẽ thực hiện các hướng dẫn của nó

Wait: Quá trình chuyển sang trạng thái chờ nếu nó cần đợi một tài nguyên chẳng hạn như chờ đầu vào của người dùng hoặc chờ tệp có sẳn

Terminated: Khi tiến trình kết thúc quá trình thực thi hoặc bị hệ điều hành chấm dứt nó sẽ được sang trạng thái kết thúc và chờ được xóa khỏi bộ nhớ chính

A diagram of a running process

Description automatically generated

Hãy xem hình ảnh này để hiểu hơn về cách một tiến trình được thực hiện

Hãy xem một VD đơn giản sau được viết bằng mã Python

A black background with white text

Description automatically generatedHãy phân tích nó nào :

Khi bạn chạy chương trình này hãy xem câu lệnh thứ nhất nó sẽ chờ người dùng nhập tên của người dùng sau thì chương trình mới quay lại trạng thái sẳn sàng sau đấy chương trình chạy tiếp hai câu lệnh tiếp theo vào kết thúc

1.4 Khối điều khiển tiến trình (PCB)

Khối điều khiển tiến trình là cấu trúc dữ liệu được Hệ điều hành duy trì cho mọi tiến trình. PCB được xác định bằng ID tiến trình số nguyên (PID). PCB lưu giữ tất cả thông tin cần thiết để theo dõi một tiến trình

1.5 Chuyển đổi giữa các tiến trình

Chuyển đổi giữa các tiến trình là cơ chế lưu trữ và khôi phục trạng thái hoặc ngữ cảnh của CPU trong khối Kiểm soát quy trình để việc thực thi quy trình có thể được tiếp tục lại từ cùng một điểm sau đó. Sử dụng kỹ thuật này, trình chuyển đổi tiến trình cho phép nhiều tiến trình chia sẻ một CPU. Chuyển đổi ngữ cảnh là một phần thiết yếu của các tính năng của hệ điều hành đa nhiệm.

A diagram of a process

Description automatically generatedKhi bộ lập lịch chuyển CPU từ việc thực thi một tiến trình này sang thực thi một tiến trình khác, trạng thái của tiến trình đang chạy sẽ được lưu vào khối điều khiển tiến trình. Sau đó, trạng thái cho quy trình chạy tiếp theo sẽ được tải từ PCB của chính nó và được sử dụng để thiết lập PC, các thanh ghi, v.v. Tại thời điểm đó, quy trình thứ hai có thể bắt đầu thực thi

Hãy xem hình này để hiểu rõ hơn về cách mà chương trình thực hiện việc chuyển đổi giữa các tiến trình( Vd này là việc chuyển đổi giữa hai tiến trình có tên là P1 và P2)

Chuyển đổi tiến trình đòi hỏi tính toán chuyên sâu vì trạng thái thanh ghi và bộ nhớ phải được lưu và khôi phục. Để tránh mất nhiều thời gian chuyển đổi tiến trình, một số hệ thống phần cứng sử dụng hai hoặc nhiều bộ thanh ghi bộ xử lý. Khi quá trình được chuyển đổi, thông tin sau sẽ được lưu trữ để sử dụng sau

Các thông tin được lưu trữ (Xác định một tiến trình)

* Bộ đêm chương trình
* Thông tin lịch trình
* Giá trị thanh ghi cơ sở và giới hạn
* Số đăng ký (Id) đang sữ dụng
* Trạng thái đã thay đổi
* Thông tin trạng thái I/O

1. – Luồng

Một luồng là một luồng thực thi thông qua mã tiến trình, với bộ đếm chương trình riêng giúp theo dõi lệnh nào sẽ thực hiện tiếp theo, các thanh ghi hệ thống chứa các biến làm việc hiện tại của nó và một ngăn xếp chứa lịch sử thực thi.

Một luồng chia sẻ với các luồng ngang hàng của nó một số thông tin như đoạn mã, đoạn dữ liệu và các tệp đang mở. Khi một luồng thay đổi một mục bộ nhớ đoạn mã, tất cả các luồng khác sẽ thấy điều đó. Một thread còn được gọi là một tiến trình nhẹ .

Các luồng cung cấp một cách để cải thiện hiệu suất ứng dụng thông qua tính song song. Các luồng thể hiện một cách tiếp cận phần mềm nhằm cải thiện hiệu suất của hệ điều hành bằng cách giảm chi phí xử lý các luồng tương đương với một quy trình cổ điển.

Mỗi luồng thuộc về chính xác một tiến trình và không có luồng nào có thể tồn tại bên ngoài một tiến trình. Mỗi luồng đại diện cho một luồng điều khiển riêng biệt. Các luồng đã được sử dụng thành công trong việc triển khai các máy chủ mạng và máy chủ web. Chúng cũng cung cấp nền tảng phù hợp để thực thi song song các ứng dụng trên bộ đa xử lý bộ nhớ dùng chung. Hình dưới đây cho thấy hoạt động của một tiến trình đơn luồng và đa luồng.

A screenshot of a computer screen

Description automatically generated

Hãy xem hình bên giữa đơn luồng và đa luồng

* 1. Tiến trình và Luồng

|  |  |
| --- | --- |
| **Tiến trình** | **Luồng** |
| Quá trình nặng và tốn nhiều tài nguyên | Luồng nhẹ, chiếm ít tài nguyên so với tiến trình |
| Chuyển đổi tiến trình cần có sự tương tác của hệ điều hàng | Chuyển đổi luồng không cần tương tác của hệ điều hàng |
| Mỗi tiến trình có một mã nhưng có tài nguyên và bộ nhớ riêng | Tất cả các luồng có thể chia sẽ cùng một tập hợp các tệp, các tiến trình con |
| Nếu một tiến trình bị chặn các tiến trình khác cũng bị chặn | Trong khi một luồng bị chặn và chờ, luồng hai trong cùng một tác vụ có thể chạy |
| Trong nhiều tiến trình, mỗi tiến trình hoạt động độc lập với tiến trình khác. | Một luồng có thể đọc, ghi hoặc thay đổi dữ liệu của luồng khác |

* 1. Ưu điểm của luồng
* Luồng giảm thiểu thời gian chuyển đổi ngữ cảnh.
* Việc sử dụng các luồng cung cấp tính đồng thời trong một tiến trình.
* Giao tiếp hiệu quả.
* Sẽ tiết kiệm hơn khi tạo và chuyển đổi ngữ cảnh các luồng.
* Các luồng cho phép sử dụng kiến ​​trúc đa bộ xử lý ở quy mô và hiệu quả cao hơn.
  1. Các loại luồng

2.3.1 Luồng cấp độ người dùng

Kernel quản lý luồng không biết đến sự tồn tại của các luồng. Thư viện luồng chứa mã để tạo và hủy các luồng, để truyền thông điệp và dữ liệu giữa các luồng, để lập lịch thực thi luồng cũng như để lưu và khôi phục bối cảnh luồng. Ứng dụng bắt đầu với một luồng duy nhất.

**Ưu điểm:**

* Chuyển đổi luồng không yêu cầu đặc quyền chế độ hạt nhân.
* Luồng cấp độ người dùng có thể chạy trên bất kỳ hệ điều hành nào.
* Lập kế hoạch có thể dành riêng cho ứng dụng trong chuỗi cấp độ người dùng.
* Luồng ở cấp độ người dùng được tạo và quản lý nhanh chóng.

**Nhược điểm:**

* Trong một hệ điều hành thông thường, hầu hết các cuộc gọi hệ thống đều bị chặn.
* Ứng dụng đa luồng không thể tận dụng được đa xử lý.

2.3.2 Luồng cấp độ kernel

Việc quản lý luồng được thực hiện bởi Kernel. Không có mã quản lý luồng trong khu vực ứng dụng. Các luồng hạt nhân được hệ điều hành hỗ trợ trực tiếp. Bất kỳ ứng dụng nào cũng có thể được lập trình để hoạt động đa luồng. Tất cả các luồng trong một ứng dụng đều được hỗ trợ trong một quy trình duy nhất.

Kernel duy trì thông tin ngữ cảnh cho toàn bộ quy trình và cho các luồng riêng lẻ trong quy trình. Việc lập lịch bằng Kernel được thực hiện trên cơ sở luồng. Kernel thực hiện việc tạo, lập lịch và quản lý luồng trong không gian Kernel. Các luồng hạt nhân thường được tạo và quản lý chậm hơn so với các luồng của người dùng.

**Ưu điểm:**

* Kernel có thể lập lịch đồng thời nhiều luồng từ cùng một tiến trình trên nhiều tiến trình.
* Nếu một luồng trong một tiến trình bị chặn, Kernel có thể lên lịch cho một luồng khác của cùng một tiến trình.
* Bản thân các luồng hạt nhân có thể đa luồng.

**Nhược điểm:**

* Các luồng hạt nhân thường được tạo và quản lý chậm hơn so với các luồng của người dùng.
* Việc chuyển quyền điều khiển từ luồng này sang luồng khác trong cùng một tiến trình yêu cầu chuyển đổi chế độ sang Kernel.

2.3.4 So sánh luồng cấp người dùng và luồng cấp kernel

|  |  |
| --- | --- |
| **Luồng cấp người dùng** | **Luồng cấp kernel** |
| Các luồng cấp người dùng được tạo và quản lý nhanh hơn | Các luồng cấp kernel được tạo và quản lý chậm hơn |
| Việc triển khai được thực hiện bằng thư viện luồng | Hệ điều hành hỗ trợ các luồng kernel |
| Có thể chạy trên mọi hệ điều hành | Dành riêng cho hệ điều hàng |
| Các ứng dụng đa luồng không tận dụng được xử lý đa luồng | Bản thân các tiến trình hạt nhân có thể là đa luồng |

* 1. Đa luồng

Đa luồng (multithreading) là khả năng cho phép một ứng dụng hoặc tiến trình thực hiện nhiều luồng thực thi đồng thời. Mỗi luồng thực thi đều có thể thực hiện một tác vụ riêng biệt, tạo ra sự song song hoạt động trên hệ thống.

**Ưu điểm :**

* Tăng hiệu năng và tiết kiệm thời gian: Do luồng dùng chung tài nguyên với tiến trình nên tạo và xóa luồng không đòi hỏi những công đoạn cấp phát, giải phóng bộ nhớ và tài nguyên nên tốn ít thời gian hơn nhiều.
* Dễ dàng chia sẻ tài nguyên và thông tin: Tài nguyên dùng chung cho phép luồng dễ dàng liên lạc với nhau.
* Tăn tính đáp ứng:là khả năng tiến trình phản ứng lại với yêu cầu của người dùng hoặc tiến trình khác
* Tận dụng được kiến trúc xử lý với nhiều CPU: Trong hệ thống nhiều CPU, các luồng có thể chạy song song trên những CPU khác nhau, nhờ vậy tăng tốc độ xử lý chung của tiến trình
* Thuận lợi cho việc tổ chức chương trình: Ví dụ, 1 chương trình thể hiện 1 vật chuyển động và phát ra âm thanh có thể tổ chức thành 2 luồng riêng, 1 luồng là đồ họa, 1 luồng là âm thanh.

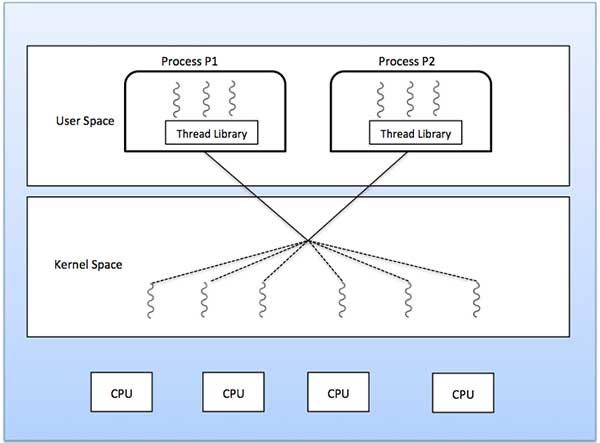
**Nhược điểm:**

* + Càng nhiều luồng thì xử lý càng phức tạp
  + Xử lý vấn đề về tranh chấp bộ nhớ, đồng bộ dữ liệu khá phức tạp
  + Cần phát hiện tránh các luồng chết (dead lock), luồng chạy mà không làm gì trong ứng dụng cả.
  1. Mô hình đa luồng

Một số hệ điều hành cung cấp tiện ích luồng cấp người dùng kết hợp và tiện ích luồng cấp hạt nhân. Solaris là một ví dụ điển hình cho phương pháp kết hợp này. Trong một hệ thống kết hợp, nhiều luồng trong cùng một ứng dụng có thể chạy song song trên nhiều bộ xử lý và lệnh gọi hệ thống chặn không nhất thiết phải chặn toàn bộ quá trình. Mô hình đa luồng có ba loại

2.5.1 Many to Many Model

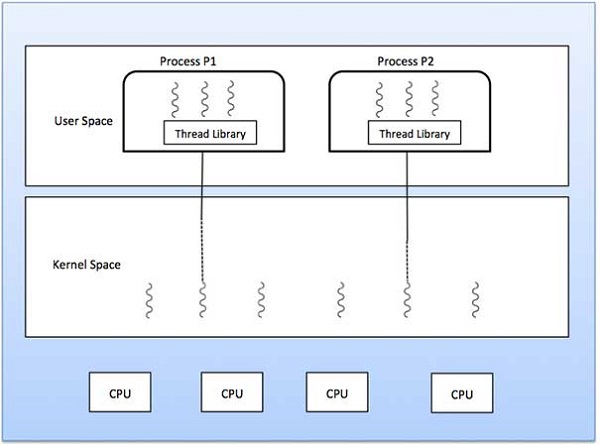
Mô hình many to many ghép bất kỳ số lượng luồng người dùng nào thành một số lượng luồng nhân bằng hoặc nhỏ hơn.



Sơ đồ sau đây hiển thị mô hình phân luồng nhiều-nhiều trong đó 6 luồng cấp người dùng đang ghép kênh với 6 luồng cấp nhân. Trong mô hình này, các nhà phát triển có thể tạo bao nhiêu luồng người dùng nếu cần và các luồng Kernel tương ứng có thể chạy song song trên một máy đa bộ xử lý. Mô hình này cung cấp độ chính xác tốt nhất về đồng thời và khi một luồng thực hiện lệnh gọi hệ thống chặn, hạt nhân có thể lên lịch cho một luồng khác

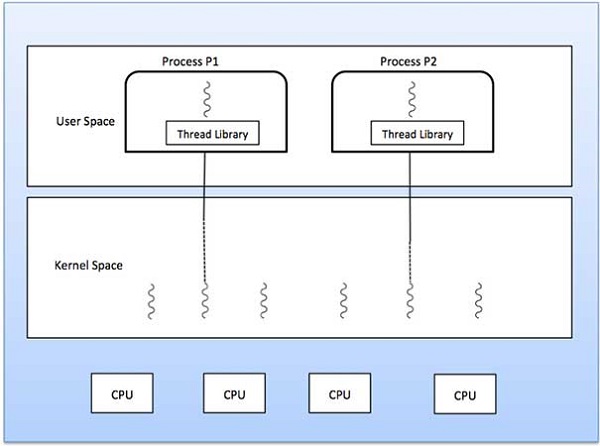
2.5.2 Many to One Model

Mô hình nhiều-một ánh xạ nhiều luồng cấp người dùng thành một luồng cấp hạt nhân. Quản lý luồng được thực hiện trong không gian người dùng bởi thư viện luồng. Khi luồng thực hiện lệnh gọi hệ thống chặn, toàn bộ quá trình sẽ bị chặn. Mỗi lần chỉ có một luồng có thể truy cập Kernel, do đó nhiều luồng không thể chạy song song trên nhiều bộ xử lý.

Nếu các thư viện luồng cấp người dùng được triển khai trong hệ điều hành theo cách mà hệ thống không hỗ trợ chúng thì các luồng hạt nhân sẽ sử dụng chế độ mối quan hệ nhiều-một.

2.5.3 One to One Model

Có mối quan hệ một-một giữa luồng cấp người dùng với luồng cấp kernel. Mô hình này cung cấp nhiều tính đồng thời hơn mô hình nhiều-một. Nó cũng cho phép một luồng khác chạy khi một luồng thực hiện lệnh gọi hệ thống chặn. Nó hỗ trợ nhiều luồng để thực thi song song trên bộ vi xử lý.

Nhược điểm của mô hình này là việc tạo luồng người dùng cần có luồng Kernel tương ứng. OS/2, windows NT và windows 2000 sử dụng mô hình quan hệ 1-1.

* 1. Các vấn đề trong xử lý tuyến

2.6.1 Đồng bộ hóa

Khi các luồng cùng truy cập vào các tài nguyên chung, việc đồng bộ hóa trở nên rất quan trọng để tránh xảy ra các tình huống đua nhau (race condition) hoặc truy cập không đồng bộ dẫn đến lỗi

2.6.2 Thời gian chờ

Đây là tình trạng mà các luồng đang chờ đợi tài nguyên khác để thực hiện tiếp công việc của mình nhưng tài nguyên đó lại đang bị giữ bởi một luồng khác. Khi đó, các luồng đó sẽ bị chờ đợi mãi mà không hoạt động được, dẫn đến tình trạng treo (hang) hoặc đứng (freeze)

2.6.3 Cấp phát và giải phóng tài nguyên

Khi các luồng cùng chia sẻ tài nguyên, việc quản lý cấp phát và giải phóng tài nguyên trở nên khó khăn và phức tạp hơn, do đó có thể dẫn đến lỗi và xảy ra hiện tượng rò rỉ bộ nhớ (memory leak)

2.6.4 Hiệu suất

Nếu không được quản lý cẩn thận, việc sử dụng xử lý tuyến có thể làm giảm hiệu suất của chương trình, do tốn nhiều tài nguyên và gây ra thời gian chờ đợi (overhead).

2.6.5 Bảo mật

Khi các luồng cùng truy cập vào dữ liệu nhạy cảm hoặc tài nguyên hệ thống, việc đảm bảo bảo mật trở nên quan trọng để tránh bị tấn công và mất dữ liệu.

2.6.6 Nguyên nhân

* Đồng bộ hóa không chính xác : Khi các luồng truy cập vào cùng một tài nguyên mà không được đồng bộ hóa đúng cách, các tình huống đua nhau (racecondition) có thể xảy ra. Điều này có thể dẫn đến lỗi và làm giảm hiệu suất của chương trình
* Sử dụng tài nguyên không đồng bộ: Khi các luồng cùng truy cập vào tài nguyên mà không được đồng bộ hóa đúng cách, các lỗi có thể xảy ra như việc sử dụng tài nguyên không tồn tại hoặc bị giải phóng mà không được thông báo cho các luồng khác
* Sử dụng cấu trúc DL không đồng bộ: Khi các luồng sử dụng cùng một cấu trúc dữ liệu mà không được đồng bộ hóa đúng cách, các lỗi có thể xảy ra như truy cập sai đến dữ liệu, mất dữ liệu hoặc dữ liệu bị thay đổi không đồng bộ.
* Quản lý tài nguyên không dúng: *Khi các luồng cùng chia sẻ tài nguyên mà không được quản lý đúng cách, các lỗi có thể xảy ra như lỗi rò rỉ bộ nhớ (memory leak), lỗi tràn bộ đệm (bufferoverflow) hoặc lỗi quản lý tài nguyên không chính xác.*.
* Lỗi thiết kế: Khi chương trình không được thiết kế để hỗ trợ xử lý tuyến, các lỗi có thể xảy ra như lỗi bảo mật, hiệu suất thấp hoặc không thể quản lý tài nguyên đúng cách
* Lỗi triển khai: Khi triển khai xử lý tuyến không đúng, các lỗi có thể xảy ra như lỗi đồng bộ hóa, lỗi tài nguyên hoặc lỗi hiệu suất
  + 1. Cách giải quyết
* Đua nhau: Sử dụng các cơ chế đồng bộ như block hoặc menitor để đảm bảo tài nguyên được truy cập đồng bộ và an toàn
* Không đồng bộ: Sử dụng các phương pháp đồng bộ hóa như mutex, semaphore,monitor để đảm bảo các tuyến được thực thi một cách đồng bộ và tránh xung đột truy cập tài nguyên
* Deadlook: Sử dụng các thuật toán phân chia tài nguyên để tránh xảy ra tình trạng tài nguyên bị giữ và không bị giải phóng
* Starvation: Sử dụng các cơ chế lập lịch để đảm bảo tất cả các tiến trình được cấp phát tài nguyên một cách công bằng
* Oversubscription: Giảm tải hệ thống băng cách giảm số tiến trình đang chạy cùng lúc hoặc tối ưu hóa các tiến trình đang chạy để giảm tải hệ thống
  1. Đơn luồng và đa luồng

|  |  |
| --- | --- |
| **Đơn luồng** | **Đa luồng** |
| Công việc được thực hiện tuần tự | Cho phép các công việc chạy song song |
| Thời gian có thể chạy nhanh hơn trong các trường hợp không cần đồng bộ hóa và không có chi phí cho việc quản lý luồng | Đa luồng thích hợp cho các tác vụ phức tạp, cần phải thực hiện đồng thời hoặc các tác vụ dòi hỏi thời gian chờ |

**Hãy bắt đầu với một ví dụ đơn luồng bằng mã Pyhon**

**Đơn luồng** là mặc định của hầu hết các ngôn ngữ lập trình trong đó có python

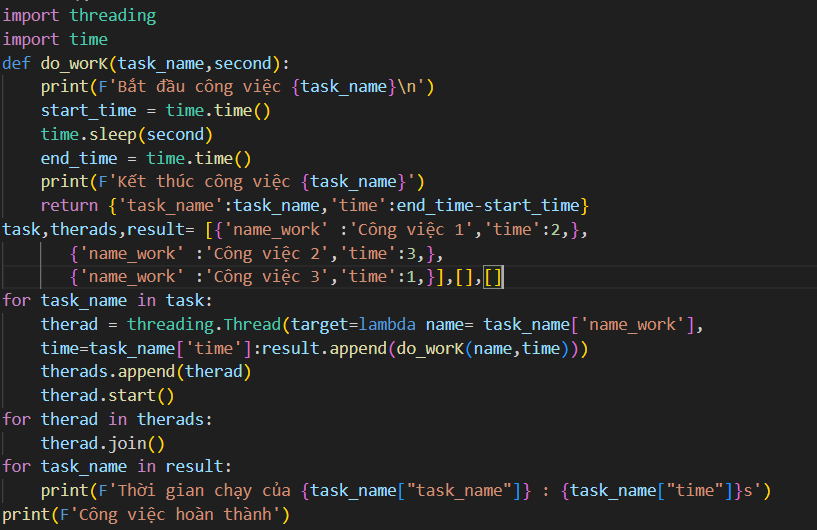
ChúnA computer screen with text and numbers

Description automatically generatedg ta khởi tạo một hàng công việc sau đấy gọi 3 lần công việc khác nhau khi chạy vì đây là đơn luồng nên sau khi công việc 1 hoàn thành công việc 2 mới được thực hiện công việc 2 hoàn thành công việc thứ 3 mới được thực hiện

A screenshot of a computer

Description automatically generatedKết quả hoàn toàn đúng với những gì mà chúng ta đã nói và thời gian thực hiện 3 công việc trên là hơn 6s

**Đa luồng:** đa luồng cấp người dùng được Python hỗ trợ và được tích hợp trong thư viện tiêu chuẩn có tên là threading (Hầu hết các ngôn ngữ lập trình khác cũng sẽ có thư viện hỗ trợ công việc đa luồng này)

Hãy xem ví dụ này

Chúng ta cũng khởi tạo 3 công việc khác nhau vì chúng ta đã sữ dụng đa luồng nên cả 3 công việc này sẽ được CPU thực hiện đồng thời nên ở đây chúng ta có thể đoán được là công việc nào có thời gian ngắn hơn được hoàn thành xong trước, Hãy chạy và xem kết quả nào

A computer screen shot of a number

Description automatically generated

Kết quả đúng với những gì chúng ta đã nói ở trên và thời gian thực hiện 3 công việc chỉ mất 3s

3 – Điều độ tiến trình

Điều độ tiến trình là sự tổ chức thực hiện các tiến trình theo một giải thuật điều độ nào đó, nhằm tránh sự xung đột giữa các tiến trình về mặt tài nguyên.Tại một thời điểm, chỉ có một tiến trình được thực hiện tại CPU,nên phải lựa chọn tiến trình nào được sử dụng CPU.Việc lựa chọn tiến trình được sử dụng CPU được gọi là điều độ Để thực hiện điều độ, cần có bộ điều độ và các thuật toán điều độ

3.1 Yêu cầu điều độ

Tài nguyên găng:Tài nguyên găng là loại tài nguyên mà tại một thời điểm chỉ có thể phục vụ cho một đối tượng .

Công việc điều độ phải thoả mãn các yêu cầu: Mỗi thời điểm chỉ có một tiến trình nằm trong đoạn găng., Không tiến trình nào được phép ở lâu vô hạn trong đoạn găng, Không tiến trình nào phải chờ vô hạn trước đoạn găng .

Đoạn găng: Là đoạn chương trình sử dụng tài nguyên găng

3.1 Mục tiêu điều độ

* Công bằng: Các tiến trình đều công bằng trong việc chia sẽ CPU
* Tuân thủ chính sách: Các chính sách đã đề ra phải được thực thi
* Cân bằng: đảm bảo các bộ phận đều phải làm việc
* Thông lượng: tối đa hóa lượng công việc được thực hiện trong mỗi giờ
* Thời gian thực hiện: Giảm thiểu thời gian từ lúc bắt đầu tới lúc kết thúc
* Tận dụng CPU: CPU phải luôn hoạt động
* Thời gian đáp ứng: Đáp ứng lại nhu cầu một cách nhanh chóng
* Sự cân xứng: Đáp lại mong chờ của người dùng
* Phải có hạn chót: Tránh mất mát dữ liệu
* Khả năng dự báo: tránh làm giảm chất lượng trong các hệ thống đa phương tiện

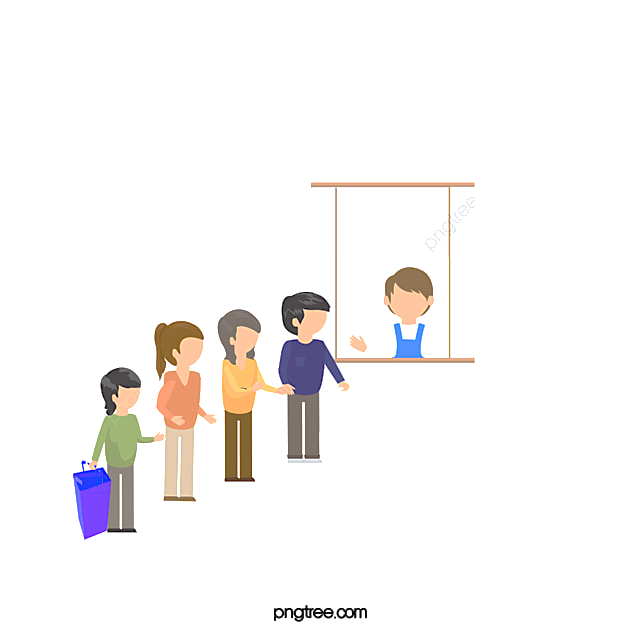
4 – Các thuật toán điều độ

4.1 First Come First Serve (FCFS) “Tới trước phục vụ trước”

FCFS được coi là thuật toán điều độ CPU đơn giản nhất. Trong thuật toán FCFS, tiến trình yêu cầu CPU trước sẽ được phân bổ vào CPU trước. Việc triển khai thuật toán FCFS được quản lý bằng hàng đợi FIFO (Vào trước ra trước). Lập kế hoạch FCFS là không được ưu tiên. Có nghĩa là khi CPU đã được phân bổ cho một tiến trình thì tiến trình đó sẽ giữ CPU cho đến khi nó được thực thi một công việc hoặc nhiệm vụ và giải phóng CPU, bằng cách yêu cầu I/O.

Đúng như tên gọi của nó thuật toán này lập lịch cho thứ tự các tiến trình đến vào một hàng đợi và sau đó lấy lần lượt ra cho CPU thực hiện thuật toán này có mặt trong cuộc sống khá nhiều

Hãy bắt đầu với 1 ví dụ:



Khi bạn xếp hàng để đợi mua một ổ bánh mì, chẳng hạn bạn bắt

buộc phải chờ những người đứng trước bạn mua xong rồi thì mới

tới lượt của bạn mua

4.1.2 Giải thuật FCFS

Tạo một hàng đợi (quen) và thêm(push) các tiến trình vào (quen) theo thứ tự đến của các tiến trình

while(quen not emty):

process = pop(quen)

if(process == break):

break

elif(have process arvied):

quen.push(all new process)

4.1.3 Ưu nhược điểm của FCFS

**Ưu điểm:**

* Dễ cài đặt
* Phù hợp cho các quy trình chạy dài hoặc khối lượng công việc không bị hạn chế nghiêm ngặt về thời gian.

**Nhược điểm**

* Thời gian chờ đợi trung bình cao.
* Không phải là một kỹ thuật lý tưởng cho các hệ thống chia sẻ thời gian
* Đây là một thuật toán lập lịch CPU không được ưu tiên, vì vậy sau khi tiến trình được phân bổ cho CPU, nó sẽ không bao giờ giải phóng CPU cho đến khi thực thi xong.

4.1.4 Ví dụ

Hãy bắt đầu với 1 ví dụ để hiểu hơn FCFS

Các công thức liên quan:

Thời gian lưu lại hệ thống của một tiến trình= Thời điểm tiến trình kết thúc sử dụng CPU – Thời điểm xuất hiện

Thời gian chờ của tiến trình = Thời gian lưu lại hệ thống của tiến trình – Thời gian tiến trình sử dụng CPU

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời điểu xuất hiện | Thời gian sử dụng CPU |
| P1 | 0 | 3 |
| P2 | 5 | 2 |
| P3 | 2 | 6 |
| P4 | 4 | 5 |
| P5 | 3 | 3 |

Hãy bắt đầu tìm ra đáp án

Tại thời điểm t = 0 trong hàng đợi chỉ có P1 đến nên CPU sẽ lấy P1 ra khỏi hàng đợi để thực thi

Tại thời điểm t = 3 sau khi P1 được thực thi xong, P3 và P5 đến hàng đợi trong đó P3 đến trước nên sau đó CPU sẽ lấy P3 ra khỏi hàng đợi để thực thi

Tại thời điểm t = 9 sau khi P3 được thực thi xong, P4, P2 đến hàng đợi nên trong hàng đợi hiện tại có P5,P4,P2 sau đó CPU sẽ lấy P5 ra khỏi hàng đợi để thực thi

Tại thời điểm t = 12 sau khi P5 thực thi xong trong hàng đợi còn P4,P2 sau đó CPU lấy P4 ra để thực thi

Tại thời điểm t = 17 sau khi P4 thực thi xong trong hàng đợi còn P2 sau đó CPU lấy P2 ra để thực thi

Tại thời điểm t = 19 sau khi P2 được thực thi trong hàng đợi không còn gì Stop

Biểu đồ Gant là biểu đồ cho chúng ta biết thứ tự thực hiện các tiến trình và thời gian nó bắt đầu và kết thúc

P2

P4

P5

P3

P1

0

19

17

12

9

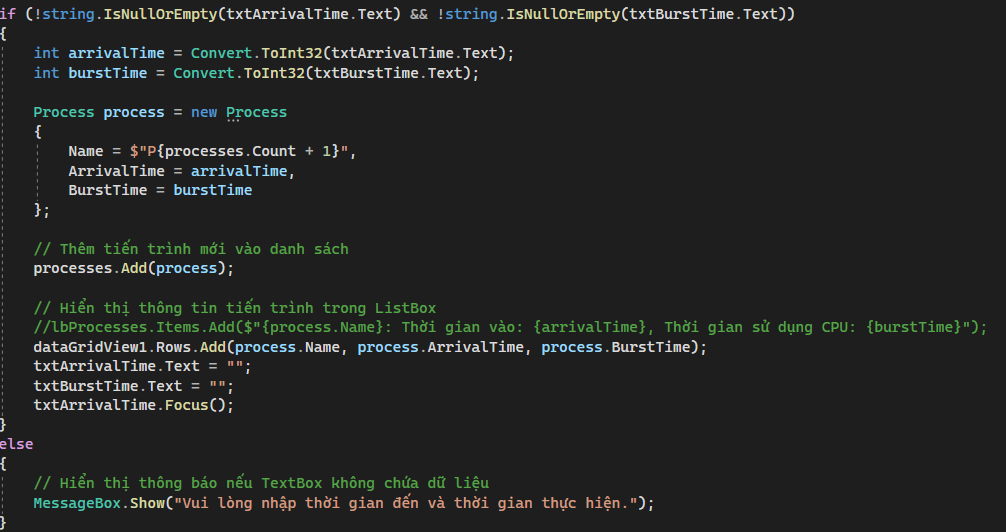
3

Từ biểu đồ gant chúng ta hãy tính thời gian chờ của các tiến trình

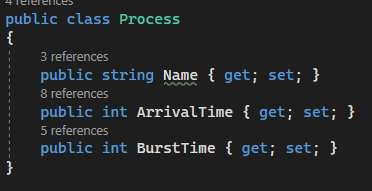
|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời gian lưu lại hệ thống | Thời gian chờ |
| P1 | 3 – 0 = 3 | 3 – 3 = 0 |
| P2 | 19 – 5 = 14 | 14 – 2 = 12 |
| P3 | 9 – 2 = 7 | 7 – 6 =1 |
| P4 | 17 – 4 = 13 | 13 – 5 = 8 |
| P5 | 12 – 3 = 9 | 9 – 3 = 6 |

Thời gian chờ trung bình = (0 + 12 + 1 + 8 + 6)/5 = 5.4

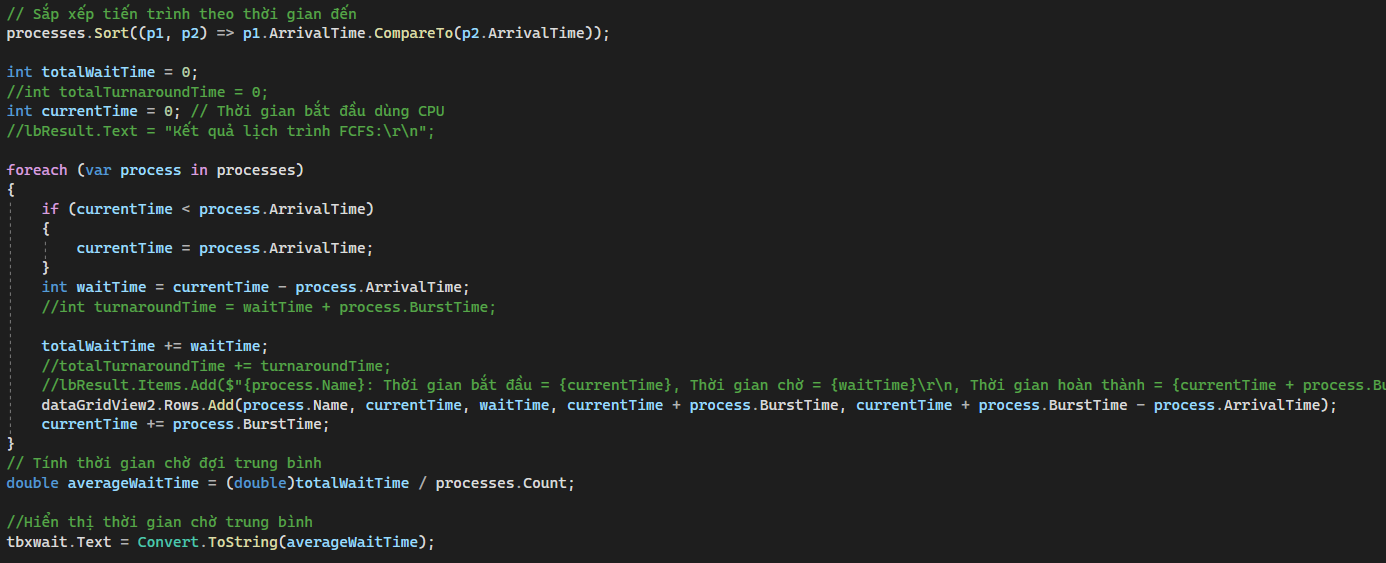
4.1.5 Cài đặt FCFS (với ngôn ngữ lập trình C# với thư viện winform)



Khởi tạo tiến trình và thêm tiến trình vào hàng đợi



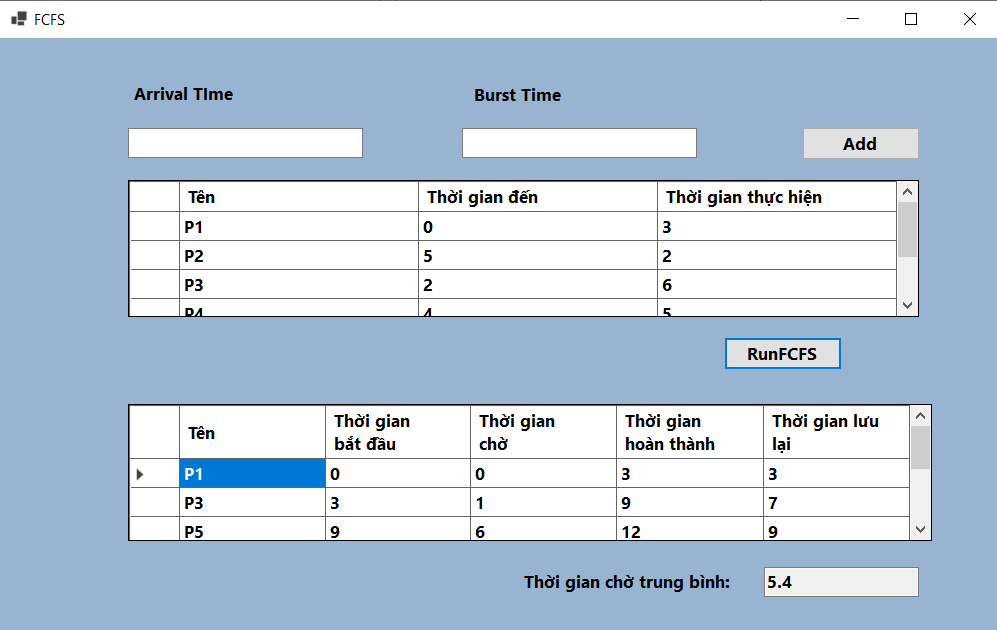
Tạo lớp tiến trình



Thuật toán(Thực hiện giải thuật)

**Kết quả sau khi chạy chương trình**

Sau khi nhập 5 tiến trình như



Chương trình sau khi chạy

Bên trên và chọn button

Run FCFS chúng ta có được

Kết quả bên dưới sau khi so

Sánh chúng ta thấy khớp với

Kết quả phân tích ở trên,

Khẳng địng tính đúng đắn của

Chương trình

4.2 RR (Round-Robin) ”Quay vòng”

Thuật toán quay vòng phân bổ cho mỗi tác vụ một phần thời gian CPU bằng nhau . Ở dạng đơn giản nhất, các tác vụ nằm trong hàng đợi vòng tròn và khi thời gian CPU được phân bổ của tác vụ hết hạn, tác vụ được đưa xuống cuối hàng đợi và tác vụ mới được lấy từ đầu hàng đợi.

Thuật toán này dựa trên thuật toán (FCFS) nó được thêm một flags để sau thời gian đấy sẽ chuyển sang tiến trình tiếp theo tránh trường hợp độc quyền CPU Thuật toán này bạn sẽ gặp trong cuộc sống rất nhiều



Vì vậy hãy bắt đầu với 1 ví dụ:

Khi bạn đăng kí học một môn học nào đấy tại trường đại học của bạn với số tín chỉ là 3 tín chẳng hạn ờ thì ở trường tôi là (45 tiết : 45h ) thì sau khi đăng kí xong bạn không thể giành liên tục 45 h đấy chỉ giành cho việc học này được đúng không, (tôi phải ăn phải ngủ phải làm các hoạt động khác nữa), nên vì vậy tôi phải chia nhỏ công việc này ra để thực hiện nó từng bước từng bước 1, thì ý tưởng của thuật toán RR này cũng như vậy

4.2.2 Giải thuật RR

Tạo một hàng đợi (quen) và thêm(push) các tiến trình vào (quen) theo thứ tự đến của các tiến trình, khởi tạo thời gian chuyển ngữ cảnh quantum =

while(quen not emty):

process = pop(quen)

if(process == break):

break

elif(time process < quatum and have process arived):

push(all new process)

elif(time process > quantum):

save\_process = save(process)

push(all new process and save\_process)

4.2.3 Ưu nhược điểm của RR

**Ưu điểm:**

* Công bằng với mỗi tiến trình khi được phân bố một lượng thời gian bằng nhau
* Phù hợp với hệ thống tương tác người dùng

**Nhược điểm**

* Đặt thời gian luân chuyển các tiến trình (quantum) quá ngắn sẽ làm tăng chi phí hoạt động và giảm hiệu suất của CPU, nhưng (quantum) quá dài có thể gây ra phản hồi kém đối với các quy trình ngắnKhông phải là một kỹ thuật lý tưởng cho các hệ thống chia sẻ thời gian
* Thời gian chờ đợi trung bình theo thuật toán RR thường dài.

4.2.4 Ví dụ

Hãy bắt đầu với 1 ví dụ để hiểu hơn RR

Các công thức liên quan:

Thời gian lưu lại hệ thống của một tiến trình= Thời điểm tiến trình kết thúc sử dụng CPU – Thời điểm xuất hiện

Thời gian chờ của tiến trình = Thời gian lưu lại hệ thống của tiến trình – Thời gian tiến trình sử dụng CPU

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời điểu xuất hiện | Thời gian sử dụng CPU |
| P1 | 0 | 3 |
| P2 | 5 | 2 |
| P3 | 2 | 6 |
| P4 | 4 | 5 |
| P5 | 3 | 3 |

Chúng ta sẽ khởi tạo quantum =4

Tại thời điểm 0 P1 đến hang đợi nên CPU lấy P1 ra để thực thi

Tại thời đi 3 sau khi P1 thực thi xong có P3 và P5 đến hàng đợi sau đó CPU lấy P3 ra để thực thi (Vì P3 đến trước)

Tại thời điểm 7 (quá hạn quantum) trong hang đợi có P4 ,P2 đến hang đợi sau đó lưu thông tin P3 và thêm vào hàng đợi Lúc này trong hàng đợi có P5,P4,P2,P3(2time) sau đó CPU lấy P5 ra để thực thi

Tại thời điểm 10 sau khi thực thi xong P5 trong hang đợi còn P4,P2,P3(2time) CPU lấy P4 ra để thực thi

Tại thời điểm 14 (quá hạn quantum) lưu thông tin P4 lại và thêm vào hàng đợi Lúc này trong hang đợi còn

P2,P3(2ime)P4(1time) CPU lấy P2 ra để thực thi

Tại thời điểm 16 P2 thực thi xong trong hang đợi còn P3(2time)P4(1time) CPU lấy P3 ra để thực thi

Tại thời điểm 18 P3 thực thi xong trong hang đợi còn P4(1time) CPU lấy P4 ra để thực thi

Tại thời điểm 19 P4 thự thi xong hàng đợi không còn gì Stop

Từ đây chúng ta sẽ có được biểu đồ Gant

P4

P3

P2

P4

P5

P3

P1

0

19

18

16

14

10

7

3

Từ biểu đồ gant chúng ta hãy tính thời gian chờ của các tiến trình

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời gian lưu lại hệ thống | Thời gian chờ |
| P1 | 3 – 0 = 3 | 3 – 3 = 0 |
| P2 | 16 – 5 = 11 | 11 – 2 = 9 |
| P3 | 18 – 2 = 16 | 16 – 6 =10 |
| P4 | 19 – 4 = 15 | 15 – 5 = 10 |
| P5 | 10 – 3 = 7 | 7 – 3 = 4 |

Thời gian chờ trung bình = (0 + 9 + 10 + 10 + 4)/5 = 6.6

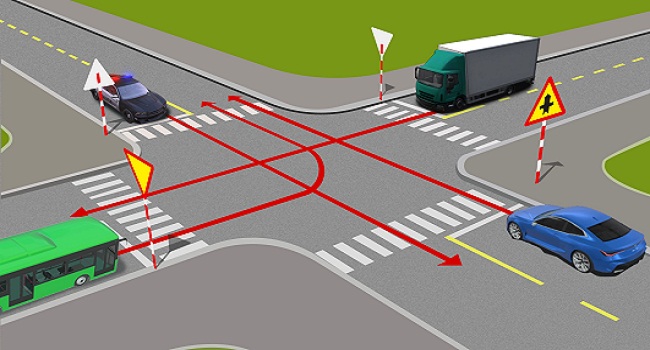
4.2.5 Cài đặt (với ngôn ngữ lập trình C# với thư viện winform)

…………………………………………………………………………………

4.3 SJF (Shortest Job First)” Ưu tiên tiến trình ngắn nhất”

SJR là một thuật toán trong đó quy trình có thời gian thực hiện nhỏ nhất được chọn cho lần thực hiện tiếp theo Nó làm giảm đáng kể thời gian chờ trung bình cho các tiến trình khác đang chờ thực thi

Thuật toán này sẽ cố gắng chọn tiến trình bé nhất hiện có trong hang đợi để ném cho CPU thực thi, Thuật toán này bạn cũng gặp rất nhiều trong cuộc sống hằng ngày Vì vậy hãy làm một ví dụ

Một ví dụ về giao thông đó là thứ tự ưu tiên giữa các xe ở Việt Nam là Xe chửa cháy, Xe quân sự, Xe cứu thương, Xe tang, Xe cậm vệ thì khi này trên đường có xe nào ưu tiên cao nhất sẽ được thông qua đi trước, đây là một quy định pháp luật nhà nước đã đặt ra, Thì cũng như vậy thuật toán SJF cũng ưu tiên dựa trên thông tin đặt ra là thời gian vì vậy sẽ ưu tiên tiến trình bé nhất thực thi trước

4.3.2 Giải thuật SJF

Tạo một hàng đợi (quen) và thêm(push) các tiến trình vào (quen) theo thứ tự đến của các tiến trình

while(quen not emty):

process = pop(find\_first\_index(min\_time\_burst(quen)))

if(process == break):

break

elif(have new process):

push(all new process)

4.3.3 Ưu nhược điểm của SJF

**Ưu điểm:**

* Ưu điểm chính của việc sử dụng thuật toán SJF là nó tối đa hóa thông lượng hệ thống, tức là số lượng tác vụ có thể được hoàn thành trong một khoảng thời gian nhất định. Bằng cách chọn các tác vụ ngắn nhất trước tiên, thuật toán SJF sẽ giảm thời gian nhàn rỗi của bộ xử lý và tăng tốc độ sử dụng

**Nhược điểm**

* Thuật toán SJF dựa trên các ước tính hoặc dự đoán về thời gian thực hiện, điều này có thể không chính xác hoặc không đáng tin cậy. Nếu ước tính sai, thuật toán SJF có thể không chọn được thứ tự nhiệm vụ tối ưu

4.3.4 Ví dụ

Hãy bắt đầu với một ví dụ để hiểu cách hoạt động của SJF

Các công thức liên quan:

Thời gian lưu lại hệ thống của một tiến trình= Thời điểm tiến trình kết thúc sử dụng CPU – Thời điểm xuất hiện

Thời gian chờ của tiến trình = Thời gian lưu lại hệ thống của tiến trình – Thời gian tiến trình sử dụng CPU

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời điểu xuất hiện | Thời gian sử dụng CPU |
| P1 | 0 | 3 |
| P2 | 5 | 2 |
| P3 | 2 | 6 |
| P4 | 4 | 5 |
| P5 | 3 | 3 |

**Start**

Tại thời điểm t = 0 P1 xuất hiện trong hang đợi CPU lấy P1 ra thực thi

Tại thời điểm t = 3 P1 thực thi xong có P3,P5 đến hàng đợi sau đấy tìm trong hang đợi tiến trình có thời gian thực thi bé nhất chúng ta được P5 sau đấy CPU lấy P5 ra thực thi

Tại thời điểm t = 6 P5 thực thi xong trong hang đơi có P3,P4,P2 sau đấy CPU lấy P2 ra thực thi (Vì P2 có thời gian thực thi bé nhất)

Tại thời điểm t = 8 P2 thực thi xong hang đợi còn P3,P4 sau đấy CPU lấy P4 ra và thưc thi (Vì P4 có thời gian thực thi bé nhất trong hang đợi)

Tại thời điểm t = 13 P4 thực thi xong hang đợi còn P3 sau đấy CPU lấy P3 ra để thực thi

Tại thời điểm t = 19 P3 thực thi xong hang đợi trống Stop

Từ đấy chung ta có biểu đồ Gant

P3

P4

P2

6

P5

P1

0

19

13

8

3

Từ biểu đồ gant chúng ta hãy tính thời gian chờ của các tiến trình

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời gian lưu lại hệ thống | Thời gian chờ |
| P1 | 3 – 0 = 3 | 3 – 3 = 0 |
| P2 | 8 – 5 = 3 | 3 – 2 = 1 |
| P3 | 19 – 2 = 17 | 17 – 6 =11 |
| P4 | 13 – 4 = 9 | 9 – 5 = 4 |
| P5 | 6 – 3 = 3 | 3 – 3 = 0 |

Thời gian chờ trung bình = (0 + 1 + 11 + 4 + 0)/5 = 3.2

4.3.5 Cài đặt (với ngôn ngữ lập trình C# với thư viện winform)

…………………………………………………………………………………

4.4 SRTN (Shortest Remaining Time Next)

“Ưu tiên tiến trình có thời gian còn lại ngắn nhất”

Thuật toán SRTN là một phương pháp lập lịch là phiên bản ưu tiên của việc lập lịch tiếp theo cho công việc có thời gian ngắn nhất. Trong thuật toán lập lịch trình này, tiến trình có khoảng thời gian nhỏ nhất còn lại cho đến khi hoàn thành được chọn để thực thi trước.

Tức là sau một đơn vị thời gian CPU sẽ xem trong hang đợi có tiến trình nào mới đến có thời gian thực thi bé hơn không nếu có lây ra thực thi Thuật toán yêu cầu việc chuyển đổi ngữ cảnh liên tục để chọn ra tiến trình tốt nhất Thuật toán này bạn cũng gặp khá nhiều trong cuộc sống vậy hãy lấy một vd cụ thể nào



Ví dụ: Chắc hẳn bạn đã từng đi xe bus rồi nhỉ (không thì thôi vậy ‘) VD bạn thấy khi lên xe bus bạn xe được yêu cầu nhường ghế cho người lớn tuổi hơn …

Lúc này khi tới mỗi trạm dứng nếu có một người lớn tuổi lên xe thì cần phải nhường chỗ cho người ấy Và khi người ấy xuống ở trạm tiếp theo và không có ai lên xe nữa thì bạn có thể ngồi rồi : cứ như vậy ở mỗi trạm tiếp theo cho đến khi hết ở đây chắc là lệnh break:

Tôi nghĩ Vd này có ít gặp bởi vì bạn không cần thực sự phải nhường ghế cho những người hơn bạn 1 vài tuổi nhưng mong bạn vẫn hiểu ý tưởng đằng sau thuật toán này

4.4.2 Giải thuật SRTN

Tạo một hàng đợi (quen) và thêm(push) các tiến trình vào (quen) theo thứ tự đến của các tiến trình

while(quen not emty):

process = pop(find\_first\_index(min\_time\_burst(quen)))

if(process == break):

break

elif(have new process):

push(all new process)

process\_min\_in\_quen = pop(find\_first\_index(min\_time\_burst(quen)))

if(process < process\_min\_in\_quen):

keep enforceable process

else:

push(save(process)), process = process\_min\_in\_quẽn

4.4.3 Ưu nhược điểm của SRTN

**Ưu điểm:**

* Thời gian chờ đợi trung bình ít
* Phản hồi tốt cho các tiến trình ngắn

**Nhược điểm**

* Khó ước lượng thời gian còn lại cần thiết để hoàn thành việc thực hiện
* Tốn kém cho việc chuyển đổi tiến trình liên tục

4.4.4 Ví dụ

Hãy bắt đầu với một ví dụ để hiểu cách hoạt động của SRTN

Các công thức liên quan:

Thời gian lưu lại hệ thống của một tiến trình= Thời điểm tiến trình kết thúc sử dụng CPU – Thời điểm xuất hiện

Thời gian chờ của tiến trình = Thời gian lưu lại hệ thống của tiến trình – Thời gian tiến trình sử dụng CPU

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời điểu xuất hiện | Thời gian sử dụng CPU |
| P1 | 0 | 3 |
| P2 | 5 | 2 |
| P3 | 2 | 6 |
| P4 | 4 | 5 |
| P5 | 3 | 3 |

**Start**

Tại thời điểm t = 0 P1 có trong hàng đợi CPU lấy P1 ra và thực thi

Tại thời điểm t = 2 P3 xuất hiện so sánh thời gian sử dụng CPU của P3 và thời gian sử dụng CPU của P1 còn lại thì thời gian sử dụng CPU còn lại của P1 bé hơn nên P1 vẫn được CPU tiếp tục thực thi

Tại thời điểm t = 3 P1 thực thi xong P5 xuất hiện lúc này trong hàng đợi còn P3, P5 sau đấy CPU lấy P5 ra và thực thi (Vì thời gian thực thi P5 bé hơn P3)

Tại thời điểm t = 4 xuất hiện P4 lúc này trong hàng đợi có P3,P4 chúng ta tìm thời gian bé nhất thực thi trong hàng đợi ta đươc P4 sau đấy so sánh thời gian thực thi P4 và thời gian thực thi của P5 còn lại (2time) P5 vẫn bé hơn nên P5 vẫn tiếp tục được CPU thực thi

Tại thời điểm t = 5 xuất hiện P2 lúc này trong hàng đợi có P3,P4,P2 chúng ta tìm thời gian bé nhất thực thi trong hàng đợi ta được P2 sau đấy so sánh với thời gian thực thi của P5 còn lại (1time) thì thời gian thực thi của P5 còn lại vẫn bé hơn nên CPU vẫn tiếp tục thực thi P5

Tại thời điểm t = 6 P5 thực thi xong trong hàng đợi còn P3,P4,P2 sau đấy CPU lấy P2 ra để thực thi (Vì P2 có thời gian thực thi ngắn nhất trong hàng đợi)

Tại thời điểm t = 8 P2 thực thi xong trong hàng đợi còn P3,P4 sau đấy CPU lấy P3 ra để thực thi (Vì P3 có thời gian thực thi bé nhất trong hàng đợi lúc nay)

Tại thời điểm t = 13 P3 thực thi xong trong hàng đợi còn P4 sau đấy CPU lấy P4 ra và thực thi

Tại thời điểm t = 19 P4 thực thi xong hàng đợi này trống Stop

Từ đây ta có biểu đồ Gant

P4

P3

P2

P5

P5

P5

3

P1

P1

0

1919

1313

6

5

8

4

2

Từ biểu đồ gant chúng ta hãy tính thời gian chờ của các tiến trình

|  |  |  |
| --- | --- | --- |
| Tiến trình | Thời gian lưu lại hệ thống | Thời gian chờ |
| P1 | 3 – 0 = 3 | 3 – 3 = 0 |
| P2 | 8 – 5 = 3 | 3 – 2 = 1 |
| P3 | 19 – 2 = 17 | 17 – 6 =11 |
| P4 | 13 – 4 = 9 | 9 – 5 = 4 |
| P5 | 6 – 3 = 3 | 3 – 3 = 0 |

Thời gian chờ trung bình = (0 + 1 + 11 + 4 + 0)/5 = 3.2

4.3.5 Cài đặt (với ngôn ngữ lập trình C# với thư viện winform)

…………………………………………………………………………………

5 – Tài liệu tham khảo

Bot AI: Chat GPT

https://www.tutorialspoint.com/operating\_system/os\_processes.htm

https://www.quora.com/What-are-processes-in-computing-What-is-their-significance-What-are-some-examples-of-these-processes

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/