TIÊU ĐỀ: BÀI TOÁN NĂM TRIẾT GIA ĂN TỐI

TỔNG QUAN VỀ ĐỀ TÀI

1. Bối cảnh và lý do thực hiện đề tài

Trong môi trường hệ điều hành đa nhiệm – đa chương – đa người sử dụng, việc chia sẻ tài nguyên cho các tiến trình của người sử dụng dùng chung là cần thiết, nhưng nếu hệ điều hành không tổ chức tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời, thì không những không mang lại hiểu quả khai thác tài nguyên của hệ thống mà còn làm hỏng dữ liệu của các ứng dụng. Và nguy hiểm hơn là việc hỏng dữ liệu này có thể hệ điều hành và ứng dụng không thể phát hiện được. Việc hỏng dữ liệu của ứng dụng có thể làm sai lệch ý nghĩa thiết kế của nó. Đây là điều mà cả hệ điều hành và người lập trình đều không mong muốn.

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên dùng chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ liệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tài nguyên dùng chung của các tiến trình. Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột với nhau khi sử dụng tài nguyên dùng chung hệ điều hành phải tổ chức cho các tiến trình này được độc quyền truy xuất sử dụng trên các tài nguyên dùng chung này.

Bài toán Năm triết gia ăn tối là bài toán kinh điển về tương tranh và chia sẻ tài nguyên dùng chung. Việc nghiên cứu bài toán giúp ta hiểu thêm về khía cạnh này của hệ điều hành.

1. Kết cấu của đồ án

* Tìm hiểu về tiến trình và các khái niệm liên quan.
* Tìm hiểu về tài nguyên găng, đoạn găng và công tác điều độ đoạn găng, áp dụng đối với bài toán Năm triết gia ăn tối.
* Tìm hiểu deadlock và cách ngăn chặn deadlock, áp dụng đối với bài toán Năm triết gia ăn tối.
* Xây dựng chương trình giải quyết bài toán Năm triết gia ăn tối.

CƠ SỞ LÝ THUYẾT

* 1. Tiến trình
     1. Khái niệm

Tiến trình là một chương trình đang xử lý, sỡ hữu một con trỏ lệnh, tập các thanh ghi và các biến. Để hoàn thành tác vụ của mình, một tiến trình có thể cần đến một số tài nguyên – như CPU, bộ nhớ chính, các tập tin và thiết bị nhập/xuất.

Cần phân biệt hai khái niệm chương trình và tiến trình. Một chương trình là một thực thể thụ động, chứa đựng các chỉ thị điều khiển máy tính để tiến hành một tác vụ nào đó ; khi cho thực hiện các chỉ thị này, chương trình chuyển thành tiến trình, là một thực thể hoạt động, với con trỏ lệnh xác định chỉ thị kế tiếp sẽ thi hành, kèm theo tập các tài nguyên phục vụ cho hoạt động của tiến trình.

* + 1. Các loại tiến trình

Có thể chia tiến trình thành hai loại: tiến trình tuần tự và tiến trình song song:

* Tiến trình tuần tự: là các tiến trình mà điểm khởi tạo của nó là điểm kết thúc của tiến trình trước đó.
* Tiến trình song song : là các tiến trình mà điểm khởi tạo của tiến trình này mằn ở thân của các tiến trình khác, tức là có thể khởi tạo một tiến trình mới khi các tiến trình trước đó chưa kết thúc.
  + 1. Tuyến

Tuyến là một thành phần của tiến trình sở hữu ngăn xếp và thực thi độc lập ngay trong mã lệnh của tiến trình. Nếu như hệ điều hành có nhiều tiến trình thì trong mỗi tiến trình bạn có thể tạo ra nhiều tuyến hoạt động song song trong hệ điều hành.

Điểm quan trọng nhất cần chú ý là một tuyến có thể làm bất cứ nhiệm vụ gì một tiến trình có thể làm. Tuy nhiên, vì một tiến trình có thể chứa nhiều tuyến, mỗi tuyến có thể coi như là một tiến trình nhỏ. Vậy, điểm khác biệt mấu chốt giữa tuyến và tiến trình là công việc mỗi cái thường phải làm.

Một điểm khác biệt nữa đó là nhiều tuyến nằm trong cùng một tiến trình dùng một không gian bộ nhớ giống nhau, trong khi tiến trình thì không. Điều này cho phép các tuyến đọc và viết cùng một kiểu cấu trúc và dữ liệu, giao tiếp dễ dàng giữa các tuyến với nhau. Giao thức giữa các tiến trình, hay còn gọi là IPC (inter-process communication) thì tương đối phức tạp bởi các dữ liệu có tính tập trung sâu hơn.

Cũng vì các tuyến dùng chung vùng nhớ và không gian địa chỉ của tiến trình, nên điểm yếu của việc dùng tuyến đó là khả năng đổ vỡ của tuyến sẽ ảnh hưởng đến tất cả các tuyến khác và toàn bộ tuyến trình đang hoạt động. Ngược lại, một tiến trình bị đổ vỡ luôn được hệ điều hành cô lập hoàn toàn không gây ảnh hưởng đến các tiến trình khác. Tiến trình có thể chạy trên nhiều máy khác nhau trong khi tuyến chỉ được thực thi trên một máy và trong một tiến trình.

* 1. Tài nguyên găng và đoạn găng
     1. Tài nguyên găng (Critical Resource)

Những tài nguyên được hệ điều hành chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung, mà có nguy cơ dẫn đến sự tranh chấp giữa các tiến trình này khi sử dụng chúng, được gọi là tài nguyên găng. Tài nguyên găng có thể là tài nguyên phần cứng hoặc tài nguyên phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên không phân chia được như là: các biến chung, các file chia sẻ…

Ví dụ sau cho thấy hậu quả của việc sử dụng tài nguyên găng trong các chương trình có các tiến trình hoạt động đồng thời:

Ví dụ: Giả sử có một chương trình, trong đó có hai tiến trình P1 và P2 hoạt động đồng thời với nhau. Tiến trình P1 phải tăng biến Count lên 1 đơn vị, tiến trình P2 tăng biến Count lên 1 đơn vị, với mục đích tăng Count lên được 2 đơn vị.

Chương trình có thể thực hiện như sau:

* Tiến trình P1 ghi nội dung biến toàn cục Count vào biến cục bộ L1
* Tiến trình P2 ghi nội dung biến toàn cọc Count vào biến cục bộ L2
* Tiến trình P1 thực hiện L1:=L1 + 1 và Count:=L1
* Tiến trình P2 thực hiện L2:=L2 + 1 và Count:=L2

Như vậy thoạt nhìn ta thấy rằng Count đã được tăng 2 đơn vị, nhưng trong thực tế có thể Count chỉ được tăng 1 đơn vị. Bởi vì nếu P1 và P2 đồng thời nhận giá trị của Count (Giả sử ban đầu Count = 4) vào L1 và L2, sau đó P1 tăng L1 lên 1 và P2 tăng L2 lên 1 (L1 = 5, L2 = 5), rồi sau đó cả P1 và P2 đồng thời ghi giá trị biến L của nó vào lại Count, thì Count chỉ tăng được 1 đơn vị, Count = 5. Đây là điều mà chương trình không mong muốn nhưng cả chương trình và hệ điều hành đều khó có thể phát hiện được.

Nguyên nhân ở trên là do 2 tiến trình P1 và P2 đồng thời truy xuất biến Count, cả khi nhận giá trị của Count, lẫn khi gán giá trị vào Count. Trong trường hợp này nếu hệ điều hành không cho phép hai tiến trình đồng thời truy xuất Count, hoặc hệ điều hành cho phéo mỗi tiến trình được độc quyền truy xuất Count trong đoạn code sau, thì lỗi sẽ không xảy ra:

P1: Begin

L1 := Count;

L1 := L1 + 1;

Count := L1;

End;

P2: Begin

L2 := Count;

L2 := L2+ 1;

Count := L2;

End;

Trong ví dụ này **tài nguyên găng** là biến Count.

* + 1. Đoạn găng (Critical Section)

Đoạn code trong các tiến trình đồng thời, có tác động đến các tài nguyên có thể trở thành tài nguyên găng được gọi là đoạn găng hay miền găng. Tức là, các đoạn code trong các chương trình dùng để truy cập đến các vùng nhớ chia sẻ, các tập tin chia sẻ được gọi là các đoạn găng.

Trong ví dụ ở mục 2.1, có 2 đoạn găng là:

{ L1 := Count và Count := L1 }

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên găng, hệ điều hành phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào (thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi (kết thúc) đoạn găng phải báo cho hệ điều hành và/hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng,… Các công tác điều khiển tiến trình thực hiện đoạn găng của hệ điều hành được gọi là điều độ tiến trình qua đoạn găng. Để công tác điều độ tiến trình qua đoạn găng được thành công, thì cần phải có sự phối hợp giữa vi xử lý, hệ điều hành và người lập trình. Vi xử lý đưa ra các chỉ thị, hệ điều hành cung cấp các công cụ để người lập trình xây dựng các sơ đồ điều độ hợp lý, để đảm bảo sự độc quyền trong việc sử dụng tài nguyên găng của các tiến trình. Tuy nhiên, dù cung cấp đầy đủ các công cụ điều độ tiến trình cho người lập trình, nhưng các hệ điều hành hiện nay đều tổ chức điều độ tiến trình ngay trong lõi (kernel), nên người lập trình ít quan tâm đến tổ chức điều độ tiến trình khi lập trình.

* + 1. Yêu cầu của công tác điều độ đoạn găng

Trước hết cần lưu ý lại rằng, nhiệm vụ điều độ tiến trình phải là sự phối hợp giữa phần cứng vi xử lý, hệ điều hành, ngôn ngữ lập trình và người lập trình; trong đó nhiệm vụ chính là của hệ điều hành và người lập trình. Cho dù nhiệm vụ điều độ là của thành phần nào, thì tất cả phải đạt được các yêu cầu sau:

* Tại một thời điểm không thể có hai tiến trình nằm trong đoạn găng.
* Nếu có nhiều tiến trình đồng thời cùng xin được vào đoạn găng thì chỉ có một tiến trình được phép vào đoạn găng, các tiến trình khác phải xếp hàng chờ trong hàng đợi.
* Tiến trình chờ ngoài đoạn găng không được ngăn cản các tiền trình khác vào đoạn găng.
* Không có tiến trình nào được phép ở lâu vô hạn trong đoạn găng và không có tiến trình phải chờ lâu mới được vào đoạn găng (chờ trong hàng đợi).
* Nếu tài nguyên găng được giải phóng thì hệ điều hành có nhiệm vụ đánh thức các tiến trình trong hàng đợi ra để tạo điều kiện cho nó vào đoạn găng.

Nguyên lý cơ bản của điều độ là tổ chức truy xuất độc quyền trên tài nguyên găng, nhưng sự bắt buộc độc quyền này còn tồn tại hai hạn chế lớn:

* Có thể dẫn tới tắc nghẽn (Deadlock) trong hệ thống.
* Các tiến trình có thể bị đói (Stravation) tài nguyên.
  1. Giải pháp điều độ tiến trình dùng Semaphore

Giải pháp này được Dijkstra đề xuất vào năm 1965. Semaphore được định nghĩa để sử dụng trong các sơ đồ điều độ như sau:

* Semaphore S là một biến nguyên, khởi gán bằng một giá trị không âm, đó là khả năng phục vụ của tài nguyên găng tương ứng với nó.
* Ứng với S có một hàng đợi F(s) để lưu các tiến trình đang bị blocked trên S.
* Chỉ có hai thao tác Down và Up được tác động đến Semaphore S. Down giảm S xuống 1 đơn vị, Up tăng S lên 1 đơn vị.
* Mỗi tiến trình trước khi vào đoạn găng thì phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình gọi Down(S) thì hệ thống sẽ thực hiện như sau: S := S – 1, nếu S >=0 thì tiến trình tiếp tục xử lý và vào đoạn găng, nếu S < 0 thì tiến trình phải vào hàng đợi để chờ cho đến khi S >= 0.
* Mỗi tiến trình ngay sau khi ra khỏi đoạn găng phải gọi Up để kiểm tra xem có tiến trình nào đang đợi ở trong hàng đợi hay không, nếu có thì đưa tiến trình trong hàng đợi vào đoạn găng.

Ở đây cần lưu ý rằng: Down và Up là các thủ tục của hệ điều hành, nên hệ điều hành đã được cài đặt cơ chế độc quyền cho nó, tức là các lệnh bên trong nó không thể tách rời nhau.

* 1. Tắc nghẽn (Deadlock) và chống tắc nghẽn
     1. Tắc nghẽn

Tắc nghẽn là hiện tượng trong hệ thống xuất hiện một tập các tiến trình, mỗi tiến trình trong tập này đều chờ được cấp tài nguyên, mà tài nguyên đó đang được một tiến trình khác trong tập này chiếm giữ. Và sự đợi này có thể kéo dài vô hạn nếu không có sự tác động bên ngoài.

Nói cách khác, mỗi tiến trình trong tập hợp đều chờ được cấp phát một tài nguyên hiện đang bị một tiến trình khác cũng ở trạng thái blocked chiếm giữ. Như vậy không có tiến trình nào có thể tiếp tục xử lý , cũng như giải phóng tài nguyên cho tiến trình khác sử dụng, tất cả các tiến trình trong tập hợp đều bị khóa vĩnh viễn.

Trong trường hợp của bài toán là khi tất cả các triết gia đều đói cùng một lúc, tất cả cùng nhấc chiếc nĩa bên tay phải của mình, và cùng chờ đợi chiếc nĩa từ hàng xóm bên tay trái dẫn đến các tiến trình bị khóa chết.

* + 1. Điều kiện hình thành tắc nghẽn

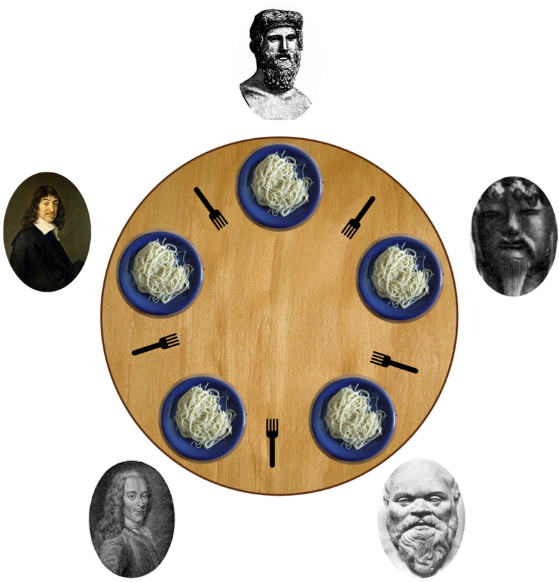
Năm 1971, Coffman đã đưa ra và chứng tỏ được rằng, nếu hệ thống tồn tại đồng thời bốn điều kiện sau đây thì hện thống sẽ xảy ra tắc nghẽn:

* Loại trừ lẫn nhau (mutual excution) hay độc quyền sử dụng: Đối với các tài nguyên không phân chia được thì tại mỗi thời điểm chỉ có một tiến trình sử dụng tài nguyên.
* Giữ và đợi (hold and wait): Một tiến trình hiện tại đang giữ tài nguyên, lại xin cấp phát thêm tài nguyên mới.
* Không ưu tiên (No preemption): Không có tài nguyên nào có thể được giải phóng từ một tiến trình đang chiến giữ nó.
* Sự tắc nghẽn có thể tồn tại với ba điều kiện trên, nhưng cũng có thể không xảy ra chỉ với ba điều kiện đó. Để chắc chắn tắc nghẽn xảy ra cần phải có điều kiện thứ tư:
* Đợi vòng tròn (Circular wait): Đây là trường hợp của ví dụ 1 mà chúng ta đã nêu ở trên. Tức là, mỗi tiến trình đang chiếm giữ tài nguyên mà tiến trình khác đang cần.
* Ba điều kiện đầu là điều kiện cần chứ không phải điều kiện đủ để xảy ra tắc nghẽn. Điều kiện thứ tư là kết quả tất yếu từ ba điều kiện đầu.
  + 1. Ngăn chặn tắc nghẽn (Deadlock Prevention)

Ngăn chặn tắc nghẽn là thiết kế một hệ thống sao cho hiện tượng tắc nghẽn bị loại trừ. Các phương thức ngăn chặn tắc nghẽn đều tập trung giải quyết bốn điều kiện gây ra tắc nghẽn, sao cho hệ thống không thể xảy ra đồng thời bốn điều kiện tắc nghẽn:

* Điều kiện này gần như khơng tránh khỏi, vì sự độc quyền là cần thiết đối với tài nguyên thuộc loại phân chia được như các biến chung, các tập tin chia sẻ, hệ điều hành cần phải hỗ trợ sự độc quyền trên các tài nguyên này. Tuy nhiên, với những tài nguyên thuộc loại khơng phân chia được hệ điều hành có thể sử dụng kỹ thuật SPOOL Smulataneous Peripheral Operation Online để tạo ra nhiều tài nguyên ảo cung cấp cho các tiến trình đồng thời.
* Đối với điều kiện giữ và đợi: Điều kiện này có thể ngăn chặn bằng cách yêu cầu tiến trình yêu cầu tất cả tài nguyên mà nó cần tại một thời điểm và tiến trình sẽ bị khoá blocked cho đến khi yêu cầu tài nguyên của nó được hệ điều hành đáp ứng. Phương pháp này khơng hiệu quả. Thứ nhất, tiến trình phải đợi trong một khoảng thời gian dài để có đủ tài nguyên mới có thẻ chuyển sang hoạt động được, trong khi tiến trình chỉ cần một số ít tài nguyên trong số đó là có thể hoạt động được, sau đó yêu cầu tiếp. Thứ hai, lãng phí tài nguyên, vì có thể tiến trình giữa nhiều tài nguyên mà chỉ đến khi sắp kết thúc tiến trình mới sử dụng, và có thể đây là những tài nguyên mà các tiến trình khác đang rất cần. Ở đây hệ điều hành có thể tổ chức phân lớp tài nguyên hệ thống. Theo đó tiến trình phải trả tài nguyên ở mức thấp mới được cấp phát tài nguyên ở cấp cao hơn.
* Đối với điều kiện No preemption: Điều kiện này có thể ngăn chặn bằng cách, khi tiến trình bị rơi vào trạng thái khố, hệ điều hành có thể thu hồi tài nguyên của tiến trình bị khố để cấp phát cho tiến trình khác và cấp lại đầy đủ tài nguyên cho tiến trình khi tiến trình được đưa ra khỏi trạng thái khóa.
* Đối với điều kiện chờ đợi vòng tròn: Điều kiện này có thể ngăn chặn bằng cách phân lớp tài nguyên của hệ thống. Theo đó, nếu một tiến trình được cấp phát tài nguyên ở lớp L, thì sau đó nó chỉ có thể yêu cầu các tài nguyên ở lớp thấp hơn lớp L.

1. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG
   1. Phân tích vấn đề
      * Bài toán đặt ra: Viết chương trình giải quyết bài toán “Năm triết gia ăn tối”. Chương trình tạo ra năm quá trình con mô phỏng hoạt động của năm triết gia. Sử dụng Semaphore để đồng bộ hoạt động của năm triết gia này.
      * Mô tả bài toán: Bài toán “Năm triết gia ăn tối” được mô tả đầy đủ như sau: Có năm triết gia cùng ngồi ăn tối trên một bàn tròn, trước mặt mỗi triết gia có một đĩa mì Ý, giữa hai triết gia thì có một chiếc nĩa. Mỗi nhà triết học đều liên tục suy nghĩ và ăn khi đói, tuy nhiên, một nhà triết học chỉ có thể ăn mì Ý khi họ cầm được chiếc nĩa bên trái và bên phải của mình. Mỗi chiếc nĩa tại mỗi thời điểm chỉ có thể được lấy bởi một triết gia và vì vậy mội triết gia chỉ có thể sử dụng chiếc nĩa khi nó không được sử dụng bởi một triết gia khác. Sau khi ăn xong, triết gia cần phải đặt hai chiếc nĩa trở lại vị trí cũ và nó trở nên sẵn có cho các triết gia khác. Giả sử việc ăn không bị giới hạn bởi số lượng mì Ý và không gian dạ dày, mỗi triết gia đều không thể biết khi nào những người khác muốn ăn hoặc suy nghĩ. Vấn đề đặt ra là làm thế nào để thiết kế một thuật toán sao cho không có triết gia nào sẽ chết đói, tức là mỗi người có thể mãi mãi luôn phiên giữa suy nghĩ và ăn khi đói.



*Hình 1. Mô tả Bài toán Năm triết gia ăn tối*

Bài toán được đặt ra bởi nhà toán học Edsger W. Dijkstra năm 1965.

* 1. Giải quyết vấn đề

Ta xem mỗi triết gia là một tiến trình, những chiếc nĩa là tài nguyên dùng chung (tài nguyên găng) cần được bảo vệ. Chương trình cần xây dựng sẽ giải quyết hai vấn đề chính, đó là Quản lý vùng găng và Ngăn chặn Deadlock.

* + 1. Quản lý vùng găng

Để quản lý vùng găng, ta xây dựng class Chopstick đại diện cho chiếc nĩa, mỗi Chopstick sẽ có một thuộc tính taken kiểu boolean để đánh dấu trạng thái của chiếc nĩa (đang được sử dụng hay không):

* taken = true: nĩa tương ứng đang được sử dụng.
* taken = false: nĩa tương ứng chưa được sử dụng.

Việc đóng/mở vùng găng được class Chopstick thể hiện qua hai phương thức synchronized put() và get():



Phương thức put() tương ứng với hành động một triết gia đặt một chiếc nĩa xuống khi ăn xong. Trong phương thức này, thuộc tính taken được gán bằng false, tức là đổi trạng thái của nó thành “chưa được sử dụng”. Biến takenBy nhằm xác định triết gia nào đang sử dụng nĩa đó (takenBy mang giá trị từ 0 đến 4 ứng với năm triết gia, và -1 nếu nó đang không được sử dụng), biến này chủ yếu phục vụ cho việc biểu diễn trên giao diện.

Ngược lại, phương thức get() tương ứng với hành động một triết gia lấy một chiếc nĩa khi thấy đói. Trong phương thức này, ta cần kiểm tra trạng thái của chiếc nĩa trước khi cho phép triết gia lấy nó, tức là nếu thuộc tính taken đang là true (nĩa đang được sử dụng) thì ta phải đợi cho đến khi taken chuyển sang giá trị false (có một triết gia khác đã đặt nĩa đó xuống). Sau đó, ta lại gán biến taken = true để đánh dấu rằng nó đang được sử dụng.

Như vậy, class Chopstick đã giải quyết được vấn đề về tài nguyên dùng chung trong bài toán, sở dĩ làm được điều này là nhờ hai hàm của hệ thống: wait() và notify(). Hiểu một cách đơn giản, wait() cho phép tiến trình được loại ra khỏi danh sách tiến trình đang hoạt động cho đến khi một tiến trình khác gọi hàm notify(), notify() thông báo và kích hoạt trở lại tiến trình đầu tiên gọi wait() trên cùng một đối tượng. Tương ứng với việc một triết gia khi cần lấy một chiếc nĩa mà nĩa đó đã được sử dụng, sẽ phải đợi đến khi nĩa đó được đặt xuống, và nếu có hai triết gia cùng muốn dùng một chiếc nĩa, triết gia cần trước sẽ được ưu tiên.

* + 1. Giải pháp ngăn chặn deadlock

Chương trình sẽ giải quyết vấn đề deadlock bằng cách xây dựng hai phương thức Wait() và Signal() trong class Philosopher (đại diện cho triết gia) sao cho vòng tròn đợi không xảy ra.



Cách giải quyết ở đây là những triết gia với số thứ tự chẵn khi đói sẽ ưu tiến lấy nĩa bên phải của mình trước, sau đó đến nĩa bên trái, những triết gia có số thứ tự lẻ thì ngược lại. Như vậy hai triết gia cạnh nhau sẽ có thứ tự lấy nĩa khác nhau, tránh được vòng tròn đợi, bởi vì nếu mỗi triết gia đều lấy nĩa bên trái trước, hoặc bên phải trước thì sẽ có nguy cơ dẫn đến vòng tròn đợi. Đây còn gọi là giải pháp bất đối xứng.

* + 1. Chương trình

Chương trình được xây dựng gồm 4 class và 1 enum:

* Class Philosopher: đại diện cho triết gia (tiến trình).
* Class Chopstick: đại diện cho nĩa (tài nguyên găng).
* Class Dinner: chịu trách nhiệm khởi tạo các class khác.
* Class DinnerGUI: quản lí giao diện.
* Enum Status: định nghĩa các trạng thái của triết gia.
  + - 1. Class Philosopher

Class gồm các thuộc tính:



Vì class Philosopher đại diện cho tiến trình nên nó kế thừa class Thread trong Java, và vì là một Thread nên những tác vụ của nó được đưa vào phương thức run():



Như vậy phương thức run() mô tả trực quan những hành động luân phiên và lặp lại của triết gia, đó là: suy nghĩ – lấy nĩa khi đói – ăn – đặt nĩa xuống khi ăn xong – suy nghĩ (lặp lại).

Hai phương thức Wait() và Signal() đã được đề cập ở mục 2.2.2. Giải pháp xử lý deadlock.

* + - 1. Class Chopstick

Class Chopstick gồm các thuộc tính:



Các thuộc tính và phương thức của Class Chopstick đã được mô tả ở mục 2.2.1. Quản lý vùng găng.

* + - 1. Class Dinner

Class Dinner dùng để khởi tạo các tiến trình và tài nguyên dùng chung, các thuộc tính và hàm dựng của class Dinner như sau:



* + - 1. Class DinnerGUI

Class DinnerGUI kế thừa JFrame, dùng để quản lý giao diện, class này có hai phương thức dùng để cập nhật giao diện khi có sự thay đổi trạng thái của triết gia hoặc của nĩa:



* + - 1. Enum Status

Enum Status định nghĩa các trạng thái của triết gia, bao gồm:

* EATING: Đang ăn
* THINKING: Đang suy nghĩ
* HUNGRY: Đang đói (lúc cần ăn nhưng chưa lấy đủ 2 nĩa)
* GETRIGHT: Đang nắm nĩa bên phải của mình, chờ nĩa bên trái
* GETLEFT: Đang nắm nĩa bên phải của mình, chờ nĩa bên trái

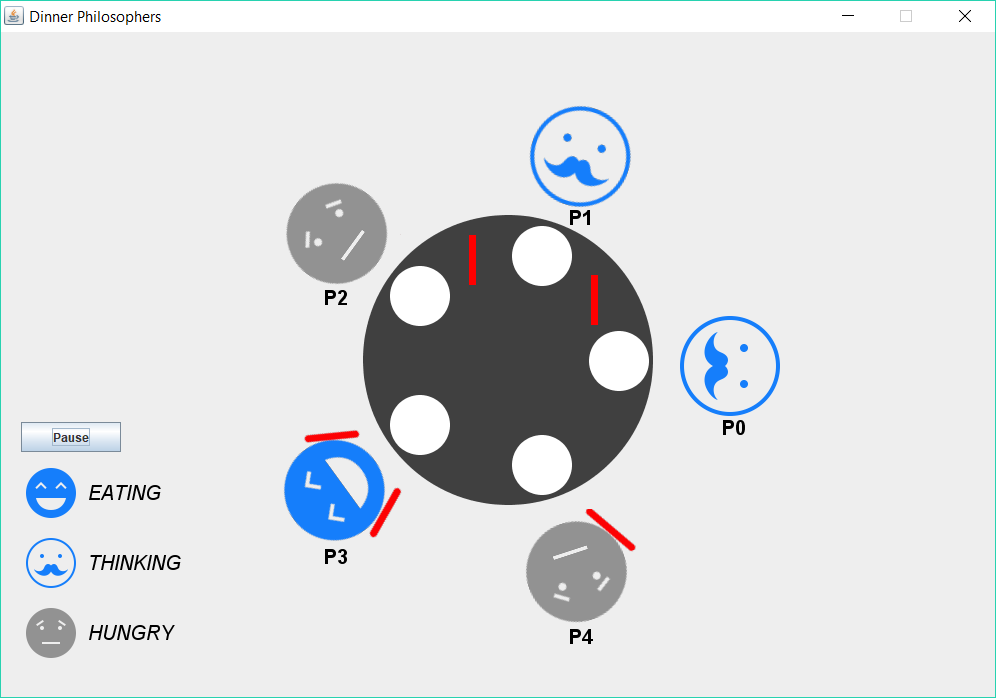
Thực chất hai trạng thái GETLEFT và GETRIGHT cũng là trạng thái lúc triết gia đói, nhưng vì mục đích hiển thị lên giao diện khác nhau, nên ta định nghĩa thêm hai trạng thái này.

1. TRIỂN KHAI   
   VÀ ĐÁNH GIÁ KẾT QUẢ
   1. Ngôn ngữ cài đặt

Chương trình được cài đặt bằng ngôn ngữ Java cơ bản và quen thuộc với sinh viên khoa Công nghệ thông tin, sử dụng công cụ Eclipse.

* 1. Kết quả chương trình

Xét một trạng thái chương trình như sau:



*Hình 2. Giao diện chương trình Bài toán Năm triết gia ăn tối*

Tại thời điểm đang xét, ta thấy:

* Triết gia số 0 và số 1 đang suy nghĩ.
* Triết gia số 3 đang ăn.
* Triết gia số 2 và số 4 đang đói.
* Triết gia số 4 đang đói và đã lấy nĩa bên phải của mình, đang chờ nĩa bên trái (đang được sử dụng bởi triết gia số 3). Còn triết gia số 2 mặc dù đang đói nhưng không lấy nĩa bên trái của mình vì phải đợi lấy chiếc nĩa bên phải trước (triết gia cố số thứ tự chẵn phải lấy nĩa bên phải của mình trước và ngược lại – giải pháp tránh deadlock).
  1. Đánh giá kết quả

Chương trình đã giải quyết được yêu cầu đặt ra của bài toán. Các kết quả thực hiện đều thỏa mãn những tiêu chí ban đầu.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

* 1. Kết luận

Đồ án đã giải quyết được vấn đề đặt ra, bao gồm việc giải quyết Bài toán Năm triết gia ăn tối và tìm hiểu những khái niệm liên quan đến bài toán đó. Công việc điều độ tiến trình của hệ điều hành là vô cùng quan trọng bởi nó giúp các tiến trình trong quá trình hoạt động tránh được các xung đột với các tiến trình khác, từ đó tránh được nguy cơ sai sót hoặc mất mác tài nguyên dùng chung.

* 1. Hướng phát triển

Chương trình tuy đã giải quyết được đầy đủ các yêu cầu cơ bản của bài toán đặt ra, nhưng hẳn vẫn còn nhiều thiếu sót và bất tiện cho người dùng. Vì vậy, chương trình có thể phát triển thêm như sau:

* Giao diện có thể được hiển thị mượt hơn khi có sự thay đổi trạng thái của các triết gia cũng như nĩa.
* Thêm một slide bar để người dùng dễ dàng chỉnh sửa thời gian trung bình của việc ăn và suy nghĩ của triết gia (trong chương trình hiện tại giá trị này là ngẫu nhiên từ 5 đến 10 giây).
* Mô tả hình ảnh triết gia sinh động hơn.