Đại Học Bách Khoa Hà Nội

Lớp Kĩ Sư Tài Năng - Công Nghệ Thông Tin K62

Các Vấn Đề Về Đồng Bộ

Ngày 1 tháng 4 năm 2019

Author

Trương Ngọc Giang
Trần Minh Hiếu
Trương Quang Khánh

Tóm tắt

Từ trước, các bài toán giải quyết vấn đề tranh trấp tài nguyên găng là một nền tảng vô cùng cơ bản trong mô hình hệ điều hành, trong các mô hình tính toán song song, và đôi khi là một số vấn đề thực tế thường gặp trong cuộc sống. Trong hệ điều hành, được thể hiện thông qua các tiến trình, các tài nguyên, luồng, ..., nhưng thực tế nó cũng được phát biểu dưới hình thức các tình huống xảy ra xung quanh, rất quen thuộc mà đôi khi ta không để ý đến. Trong báo cáo này sẽ là các bài toán đều có đặc điểm chung gồm nhiều loại tiến trình yêu cầu tài nguyên găng và tài nguyên găng sẽ được cung cấp chỉ khi ta có mô hình tổ hợp tiến trình - tài chỉ nguyên phù hợp. Phần 1 sẽ trình bày về vấn đề người hút thuốc lá, là một bài toán thể hiện cho tình huống phải giải quyết với tình trạng deadlock, khi rằng buộc bài toán được thực hiện trong môi trường hệ điều hành, tức là chỉ có thể thao tác với code của các tiến trình người dùng mà không được là hệ điều hành. Phần 2 nói về bài toán người qua sông, một bài toán có thể xảy ra tình trạng livelock.

Mục lục

1 Vấn đề người hút thuốc lá **2**
- 1.1 Đặt vấn đề . . . 2
- 1.2 Mô hình code của đại lý . . . 2
- 1.3 Lời giải vấn đề với các câu lệnh rẽ nhánh . . . 3
- 1.4 Lời giải với dãy đèn báo Semaphore [2] . . . 4
- 1.5 Trường hợp đại lý không chờ đợi tín hiệu từ người hút . . . 5
- 1.6 Nhận xét . . . 5

2 Vấn đề Qua sông **6**
- 2.1 Đặt vấn đề . . . 6
- 2.2 Mô hình code . . . 6
- 2.3 Lời giải trên blog của Marcelo Bytes[4] . . . 6
- 2.4 Lời giải của tác giả Allen B.Downey . . . 7

1 Vấn đề người hút thuốc lá

Bài toán về người hút thuốc lá được đưa ra bởi Suhas Patil [1].

1.1 Đặt vấn đề

> Có 3 người hút thuốc là và một đại lý sản xuất thuốc lá. Để hút một điếu thuốc lá cần 3 thành phần là thuốc, giấy và diêm. Đại lý thì sở hữu 3 thành phần trên, còn mỗi người hút thuốc chỉ có duy nhất một thành phần, không ai giống ai, nên đặt tên là người có thuốc, người có giấy và người có diêm, giả sử sự sở hữu ở đây là có vô hạn. Người hút thuốc chỉ thực hiện hai công việc đó là lấy thành phần và hút thuốc. Công việc của đại lý là chọn ngẫu nhiên hai thành phần khác nhau, đưa chúng ra bàn trưng bày, và tuỳ thuộc vào hai thành phần đó mà người hút thuốc trưng bày, có thành phần bổ sung sẽ lấy nó và tiến hành hút thuốc. Ví dụ như đại lý chọn ra thuốc và giấy, thì khi đó người có diêm sẽ lấy các thành phần của đại lý, ra hiệu rằng anh ta đã lấy đủ thành phần cho đại lý biết và cả hai tiếp tục công việc của mình.

Trong bài toán trên, đại lý giống như một hệ điều hành phân phối tài nguyên, còn các người hút thuốc là các tiến trình yêu cầu tài nguyên. Khi tài nguyên khả dụng thì sẽ phân phối cho các tiến trình cần nó, nhưng sẽ tránh trường hợp phân phối cho tiến trình không thể tiến hành sử dụng tài nguyên đó. Với giả thuyết trên, có 3 phiên bản được đưa ra như là một rằng buộc khi giải quyết vấn đề này:

- The Impossible Version: Không được tác động đến code của đại lý. Điều đó cũng giống như việc không thể thay đổi code của hệ điều hành mỗi khi có một được cài đặt. Ngoài ra cũng không được sử dụng câu lệnh điều kiện, hay như là dùng một dãy semaphores. Với rằng buộc này thì vấn đề không thể giải quyết.
- The Interesting Version: Bài toán sẽ trở lên thú vị và có thể giải quyết khi bỏ qua điều kiện thứ hai, tức là rằng buộc duy nhất là không được thay đổi code của đại lý.
- The Trivial Version: Trong trường hợp khác, bài toán được phép giải quyết với điều kiện rằng đại lý sẽ lựa chọn người hút thuốc phù hợp với sản phẩm mình làm ra rồi phân phối cho người đó. Khi đó, vấn để trở lên tầm thường, các thành phần và việc hút thuốc trở lên ít liên quan. Ngoài ra, trong thực tế, rất khó khăn khi bắt hệ điều hành phải biết trước về các tiến trình và tài nguyên chúng yêu cầu.

Sau đây là hướng tiếp cận của trường hợp chỉ bị rằng buộc rằng không thể thay đổi code của đại lý.

1.2 Mô hình code của đại lý

Trước tiên, ta sẽ xây dựng mô hình hoạt động đơn giản của các tiến trình.

Thiết lập đèn báo cho đại lý:

```
agentSem = Semaphore (1)
tobaco = Semaphore (0)
paper = Semaphore (0)
match = Semaphore (0)
```

Công việc của đại lý là tạo ra hai thành phần khác nhau, đợi người hút thuốc thích hợp đến lấy, rồi nhận tín hiệu từ người hút thuốc rằng đã lấy, lại tiếp tục tạo hai thành phần ngẫu nhiên, lặp đi lặp lại vô hạn.

Như vậy, sẽ có 3 trường hợp xảy ra, và đoạn code tương ứng là:

Tiến trình A

```
agentSem.wait ( )
tobaco.signal ( )
paper.signal ( )
```

Tiến trình B

```
agentSem.wait ( )
paper.signal ( )
match.signal ( )
```

Tiến trình C

```
agentSem.wait ( )
tobaco.signal ( )
match.signal ( )
```

Người có thuốc

```
paper.wait ( )
match.wait ( )
agentSem.signal ( )
```

Người có giấy

```
tobaco.wait ( )
match.wait ( )
agentSem.signal ( )
```

Người có diêm

```
paper.wait ( )
tobaco.wait ( )
agentSem.signal ( )
```

Với mô hình đơn giản như vậy, ta có thể hình dung được tình huống deadlock có thể xảy ra.

Ví dụ như trong trường hợp đại lý bật đèn báo có thuốc thì người có diêm đến lấy và đợi đại lý cung cấp giấy, sau đó đại lý bật đèn báo có diêm, thì có thể lúc đó người có thuốc đến lấy, thì người có thuốc lại đợi đại lý bật đèn báo có giấy, còn đại lý thì đợi tín hiệu hoàn thành từ khách hàng rồi mới tiếp tục công việc. Như vậy sẽ xảy ra deadlock.

1.3 Lời giải vấn đề với các câu lệnh rẽ nhánh

Theo phân tích trên thì thấy rằng tình huống deadlock xảy ra khi đại lý cho phép người hút lấy thành phần mà không quan tâm người đến việc người hút có thể tiến hành sử dụng sản phẩm được ngay sau đó hay không. Như vậy, để giải quyết vấn đề này, ta có thể cài đặt hệ thống phân phối, sẽ là trung gian giữa đại lý và người hút.

Công việc của hệ thống phân phối này đó là kiểm tra các thành phần trên bàn phù hợp với người nào, sau đó báo cho người đó biết. Các người hút chỉ đến lấy khi được hệ thống phân phối báo.

Câu trúc dữ liệu của nhà phân phối được khởi tạo gồm:

```
isTobaco = isPaper = isMatch = false
tobacoSem = semaphore (0)
paperSem = semaphore (0)
matchSem = semaphore (0)
```

Trong đó, các biến boolean thể hiện rằng thành phần tương ứng có sẵn trên bàn hay không.

Các semaphore có chức năng đánh thức người hút tương ứng với các thành phần thuốc ở trên bàn. Dưới đây là code cho nhà phân phối trường hợp đại lý đưa ra thuốc:

```
tobaco.wait( )
mutex.wait( )
    if isPaper :
       isPaper = false
       matchSem.signal( )
    else if isMatch :
       isMatch = false
       paperSem.signal( )
    else :
       isTobaco = true
mutex.signal( )
```

Nhà phân phối sẽ nhận sản phẩm từ đại lý, đặt lên bàn và kiểm tra xem trên bàn đã đủ hai thành phần chưa, nếu đủ rồi thì thông báo cho người hút thuốc tương ứng.

Dưới đây là đoạn code của người có thuốc:

```
tobaccoSem.wait ( )
makeCigarette ( )
agentSem.signal ( )
smoke ( )
```

Ngoài cách làm trên, dưới đây là một công cụ khác để kiểm tra các sản phẩm đang có mặt trên bàn.

1.4 Lời giải với dãy đèn báo Semaphore [2]

Vấn đề của bài toán này chính là ở chỗ khi chưa xác nhận được hai thành phần mà đại lý cung cấp phù hợp với người hút nào thì họ đã đến lấy thành phần, dẫn đến tình huống người hút lấy thành phần xong không thể sử dụng.

Giải pháp vẫn là chỉ cho người hút lấy khi đã xác định được hai thành phần, nhưng ta sẽ sử dụng dãy đèn báo semaphore.

Trước tiên, khởi tạo:

```
mutex = semaphore (1)
integer t = 0
senaphore array S[ 1 : 6 ] ; """ Khởi tạo các đèn báo đều bằng không """
```

Ý tưởng là bằng việc sử dụng biến dùng chung t để xác định trên bàn đang có những thành phần nào.

Ta sẽ có mô hình code các tiến trình đó như sau:

Khi đại lý đưa ra thuốc

```
tobaco.wait ( )          paper.wait ( )          match.wait ( )
mutex.wait ( )           mutex.wait ( )          mutex.wait ( )
t = t + 1                t = t + 2               t = t + 4
s[t].signal              s[t].signal ( )         s[t].signal ( )
mutex. signal ( )        mutex.signal ( )        mutex.signal ( )
```

Với cách cài đặt này thì:

- t = 3 , giao 2 sản phẩm trên bàn cho người có diêm.
- t = 5 , giao 2 sản phẩm trên bàn cho người có giấy.

- t = 6 , giao 2 sản phẩm trên bàn cho người có thuốc.

Sau khi người hút nhận được thành phần, đặt lại giá trị t = 0, báo hiệu cho đại lý, sau đó hút thuốc. Dưới đây là code cho người có diêm, người có giấy và người có thuốc

```
s[3].wait ( )              s[5].wait ( )              s[6].wait ( )
t = 0                      t = 0                      t = 0
makeCigarette ( )          makeCigarette ( )          makeCigarette ( )
agentSem.signal ( )        agentSem.signal ( )        agentSem.signal ( )
smoke ( )                  smoke ( )                  smoke ( )
```

Hai lời giải trên là dành cho trường hợp khi mà đại lý chờ đợi người hút đến lấy các thành phần rồi thông báo rồi thì mới tiếp tục đưa ra các thành phần tiếp theo. Sẽ ra sao nếu như đại lý không chờ đợi mà tiếp tục đưa ra các thành phần.

Khi đó, bài toán sẽ trở lên khó khăn hơn, vì lúc đó đại lý có thể phục vụ cho nhiều hơn một người cùng một thời điểm. Ngoài việc xác định xem thành phần nào có mặt trên bàn, ta phải xác nhận thêm cả về số lượng hiện có. Để đơn giản nhất, ta sẽ sử dụng các câu lệnh điều kiện thay vì sử dụng một dãy đèn báo semaphore.

1.5 Trường hợp đại lý không chờ đợi tín hiệu từ người hút

Trong lời giải trước, ta sử dụng các biến boolean để đếm các thành phần có trên bàn, do mỗi thành phần tối đa chỉ có một. Khi điều kiện thay đổi như trên, để đếm được, ta sẽ dùng biến nguyên. Công việc của nhà phân phối vẫn không thay đổi, tại một thời điểm chỉ cho một người lấy, dựa vào số lượng sản phẩm trên bàn để thông báo đến người phù hợp.

```
numTobaco = numPaper = numMatch = 0
tobacoSem = semaphore (0)
paperSem = semaphore (0)
matchSem = semaphore (0)
```

Dưới đây là code cho nhà phân phối trong trường hợp đại lý đưa ra thuốc:

```
  tobacco.wait( )
mutex.wait ( )
   if numPaper :
      numPaper -= 1
      matchSem . signal ( )
   elif numMatch :
      numMatch -= 1
      paperSem . signal ( )
   else :
      numTobacco += 1
mutex.signal ( )
```

1.6 Nhận xét

Trong vấn đề trên, ta đã sử dụng các nhà phân phối như là các phương thức trung gian giữa đại lý và người hút có nhiệm vụ chính là đưa sản phẩm từ đại lý ra bàn trừng bầy rồi tìm kiếm người hút thích hợp. Các sản phẩm được đặt trên bản, vàn tại một thời điểm chỉ có một nhà phân phối là được kiểm tra số lượng sản phẩm trên bàn, nêu thấy có thành phần phù hợp thì lấy nó và đưa cho người hút, còn không thì đặt thành phần mình đang giữ xuống bàn.

2 Vấn đề Qua sông

Bài toán Qua sông (River Crossing Problem) được trích dẫn từ bộ bài tập của Anthony Joseph tại đại học U.C. Berkeley, tuy nhiên không rõ ông có phải là tác giả của bài toán này hay không [3]. Bài toán này gần giống bài toán H2O, tuy nhiên khác ở chỗ là barrier chỉ cho phép một số tổ hợp tiến trình nhất định được phép qua.

2.1 Đặt vấn đề

> Ở một nơi gần Redmond, Washington có một con đò được sử dụng bởi cả các hacker Linux và nhân viên (nông nô) của Microsoft để qua sông. Con đò có thể chở được đúng 4 người; nó sẽ không rời bờ nếu như chưa đủ người hoặc quá người. Để đảm bảo sự an toàn của các hành khách, chúng ta không thể để một hacker ngồi riêng đò với ba nhân viên, hoặc một nhân viên ngồi riêng với ba hacker. Bất cứ một tổ hợp nào khác đều là hợp lệ.
> Khi một tiến trình "lên đò", nó cần phải gọi hàm một hàm board(). Bạn cần phải đảm bảo cả bốn tiến trình trên cùng một chuyến đò đều phải gọi hàm board() này trước bốn tiến trình của chuyến đò tiếp theo.
> Khi cả bốn tiến trình đã gọi hàm board(), chính xác một tiến trình phải gọi hàm rowBoat(), thể hiện rằng tiến trình này sẽ là tiến trình lái đò. Không quan trọng tiến trình nào là tiến trình lái đò, miễn là một trong số chúng phải gọi hàm này.

2.2 Mô hình code

Tác giả của bài toán đã đưa ra mô hình sau trong lời giải của mình:

```
barrier = Barrier(4)
mutex = Semaphore(1)
hackers = 0
serfs = 0
hackerQueue = Semaphore(0)
serfQueue = Semaphore(0)
local isCaptain = False
```

Trong đó:

- hackers và serifs là biến đếm số lượng hacker và nhân viên đang chờ được lên đò. Vì chúng đều được bảo vệ bởi mutex, ta có thể kiểm tra giá trị của cả hai hàm mà không phải lo lắng về các tiến trình khác cập nhật giữa chừng.
- hackerQueue và serfQueue cho phép ta kiểm soát số lượng hacker và nhân viên được qua. Barrier đảm bảo rằng cả bốn tiến trình đã gọi hàm board() trước khi gọi hàm rowBoat().
- isCaptain là một biến địa phương cho biết tiến trình nào phải gọi rowBoard().

2.3 Lời giải trên blog của Marcelo Bytes[4]

Ý tưởng chủ đạo của lời giải này xoay quanh trạng thái có 3 tiến trình đã lên đò. Khi đó:

- Nếu có hai hacker và một nhân viên lên đò, chỉ có nhân viên mới được lên đò.
- Nếu có ba hacker, chỉ có hacker mới được lên đò.
- Nếu có ba nhân viên, chỉ có nhân viên mới được lên đò.
- Nếu có hai nhân viên và một hacker, chỉ có hacker mới được lên đò.

Ta sẽ sử dụng hai thủ tục độc lập, tạm gọi là hackerGenerator() và serfGenerator() để khởi tạo tiến trình thuộc loại tương ứng. Khi đó trong thủ tục chính, khi xuất hiện trạng thái 3 người lên đò, ta có thể chặn một trong hai thủ tục này để chỉ cho phép thủ tục thuộc đúng loại được lên đò.

Đoạn code sau xử lý việc lên đò của hacker. Đối với nhân viên ta cũng xử lý tương tự.

```
hackers += 1
if hackers + serfs == 3:
    if hackers == 2:
        # Block hackerGenerator()
    if hackers == 3:
        # Block serfGenerator()
    if serfs == 2:
        # Block serfGenerator()
    if serfs == 3:
# Block hackerGenerator()
if hackers + serifs == 4:
    isCaptain = True

board()
barrier.wait()

if isCaptain:
    rowBoat()
    # Thả tất cả các block hiện tại
```

Nhận xét: Lời giải này cung cấp một hướng tiếp cận mới cho bài toán Qua Sông, song vẫn còn hạn chế - Trong trường hợp có 2 hacker và 1 nhân viên đã lên đò, việc block hackerGenerator() sẽ khiến cho chương trình phải chờ đợi cho tới khi một tiến trình nhân viên mới xuất hiện. Nếu trong một thời gian dài chỉ có tiến trình hacker xuất hiện, chương trình sẽ rơi vào trạng thái livelock.

2.4 Lời giải của tác giả Allen B.Downey

Ý tưởng chủ đạo của lời giải này là khi mỗi một tiến trình xuất hiện, ta cập nhật bộ đếm tương ứng lên một và kiểm tra xem số lượng hiện tại đã đủ để tạo thành một tổ hợp hoàn chỉnh hợp lệ hay chưa.

Đoạn code sau xử lý việc lên đò của hacker. Đối với nhân viên ta cũng xử lý tương tự.

```
mutex.wait()
    hackers += 1
    if hackers == 4:
        hackerQueue.signal(4)
        hackers = 0
        isCaptain = True
    elif hackers == 2 and serfs >= 2:
        hackerQueue.signal(2)
        serfQueue.signal(2)
        serfs -= 2
        hackers = 0
        isCaptain = True
    else:
        mutex.signal() # Tiến trình lái đò tiếp tục giữ mutex

hackerQueue.wait()

board()
barrier.wait()

if isCaptain:
    rowBoat()
    mutex.signal() # Tiến trình lái đò có thể bỏ mutex
```

Khi mỗi tiến trình được mutex cho qua, nó kiểm tra xem số lượng tiến trình đang đợi có đủ tạo thành một tổ hợp hoàn chỉnh hay không. Nếu có, nó ra tín hiệu phù hợp cho các tiến trình đang đợi, tự tuyên bố là thuyền trưởng (người lái đò) và tiếp tục giữ mutex để ngăn các tiến trình khác lên đò cho tới khi chuyến đò hiện tại đã khởi hành.

Barrier sẽ lưu lại số lượng tiến trình đã gọi hàm board(). Khi đã đủ số lượng tiến trình, thuyền trưởng có thể gọi hàm rowBoard() và bỏ mutex để cho phép các tiến trình khác lên chuyến đò tiếp theo.

Lời giải này không làm nảy sinh vấn đề như lời giải đầu, vì nó không cho tiến trình nào lên đò cho tới khi chắc chắn có thể hình thành một tổ hợp hợp lệ.đầu

Tài liệu

[1] Allen B.Downey, *The Little Book Of Semaphores version 2.2.1*, chapter 4, second edition, 2016

[2] Parnas, David. (1975). On a solution to the cigarette smoker's problem (without conditional statements). Communications of the ACM. 18. 181-183. 10.1145/360680.360709.

[3] Allen B.Downey, *The Little Book Of Semaphores version 2.2.1*, chapter 5, second edition, 2016

[4] River crossing problem, *https://blog.ksub.org.*