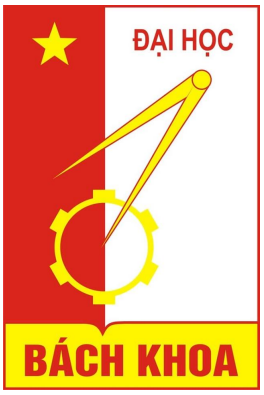
Đại học Bách Khoa Hà Nội

Viện Công nghệ thông tin và Truyền thông




# Báo cáo tìm hiểu các bài toán về điều độ tiến trình

*Giảng viên hướng dẫn:*
TS. Phạm Đăng Hải

*Sinh viên:*
Đặng Lâm San, MSSV: 20170111
Hoàng Minh Tân, MSSV: 20170112

Hà Nội - Ngày 13 tháng 4 năm 2019

# Mục lục

**1 Lời mở đầu** **3**

**2 Các bài toán đồng bộ hóa** **4**
2.1 Bài toán người hút thuốc lá . . . . . 4
2.1.1 Vấn đề . . . . . 4
2.1.2 Giải pháp . . . . . 4
2.2 Bài toán xe buýt Senate . . . . . 5
2.2.1 Vấn đề . . . . . 5
2.2.2 Giải pháp . . . . . 5
2.3 Bài toán ông già Noel . . . . . 6
2.3.1 Vấn đề . . . . . 6
2.3.2 Giải pháp . . . . . 7
2.4 Bài toán băng qua sông . . . . . 9
2.4.1 Phát biểu bài toán . . . . . 9
2.4.2 Phân tích bài toán . . . . . 9
2.4.3 Giải pháp cho bài toán . . . . . 9
2.4.4 Các vấn đề còn tồn tại . . . . . 10
2.5 Bài toán về phòng cho bữa tiệc . . . . . 10
2.5.1 Phát biểu bài toán . . . . . 10
2.5.2 Phân tích bài toán . . . . . 11
2.5.3 Giải pháp cho bài toán . . . . . 11
2.6 Bài toán bể bơi . . . . . 13
2.6.1 Phát biểu bài toán . . . . . 13

**3 Tổng kết** **14**

**4 Reference** **14**

**References** **14**

# 1 Lời mở đầu

Khi các tiến trình, luồng thực thi đồng thời và chia sẻ dữ liệu thì xảy ra vấn đề dùng chung về tài nguyên (shared memory, file, ... ). Nếu không có sự kiểm soát khi truy cập các dữ liệu, tài nguyên được chia sẻ thì có thể gây ra các trường hợp không nhất quán dữ liệu, hệ thống bế tắc, ... Vì vậy để duy trì tính ổn định, hệ thống cần có cơ chế đảm bảo sự thực thi có trật tự của các tiến trình, luồng đồng thời.
Vì vậy, một lớp các bài toán kinh điển về đồng bộ hóa tiến trình được đề xuất. Các bài toán kinh điển này thường xuất hiện trong đời sống bình thường, vì vậy phát biểu bài toán rất rõ ràng và dễ hiểu. Lý do chúng tôi quan tâm đến những bài toán này là vì chúng đều thuộc về lớp bài toán về đa tiến trình của hệ điều hành mà cần phải giải quyết. Đối với mỗi bài toán, chúng tôi đưa ra cách phát biểu bài toán, một số giải pháp cụ thể. Sau đây là một số bài toán mà chúng tôi sẽ đề cập.

- Bài toán người hút thuốc lá
- Bài toán xe buýt Senate
- Bài toán ông già Noel
- Bài toán băng qua sông
- Bài toán về phòng cho bữa tiệc
- Bài toán bể bơi

# 2 Các bài toán đồng bộ hóa

## 2.1 Bài toán người hút thuốc lá

### 2.1.1 Vấn đề

Bài toán người hút thuốc lá được đưa ra lần đầu tiên bởi Suhas Patil.
Có 4 người, trong đó có 3 người hút thuốc (A, B, C) và 1 người cung cấp nguyên liệu để quấn thuốc (agent). Những người hút thuốc thực hiện công việc lặp đi lặp lại: đợi lấy nguyên liệu, quấn thuốc và hút thuốc. Các nguyên liệu bao gồm : lá thuốc (tobacco), giấy (paper) và diêm (match). Phải có đủ 3 loại nguyên liệu mới quấn được thuốc.
Giả sử người cung cấp có không giới hạn nguyên liệu và 3 người hút có không giớ hạn 1 loại nguyên liệu (ví dụ A có lá thuốc, B có giấy và C có diêm). Quá trình sau được lặp đi lặp lại: người cung cấp sẽ chọn ngẫu nhiên 2 loại nguyên liệu để đưa cho những người hút. Dựa vào 2 loại nguyên liệu được chọn, sẽ chỉ có 1 người phù hợp để lấy 2 nguyên liệu và thục hiện tiếp công việc (quấn và hút). Ví dụ người cung cấp chọn lá thuốc và diêm thì người hút có sẵn giấy (B)sẽ lấy 2 nguyên liệu, tạo 1 điếu thuốc và ra tín hiệu cho người cung cấp.
Ở đây người cung cấp đại diện cho 1 hệ điều hành với nguyên liệu là các tài nguyên có sẵn, còn những người hút giống như các ứng dụng đang cần tài nguyên. Vấn đề là đảm bảo nếu tài nguyên sẵn có đủ cho ứng dụng nào thì ứng dụng đó phải được thực thi.

### 2.1.2 Giải pháp

Giải pháp được đề xuất bởi Parnas: dùng 3 luồng hỗ trợ gọi là "pusher" để phản hồi lại tín hiệu từ người cung cấp, theo dõi các tài nguyên sẵn có và đánh thức người hút thích hợp.
Các biến và các semaphore như sau:

```
1        isTobacco = isPaper = isMatch = False
2        tobaccoSem = Semaphore(0)
3        paperSem = Semaphore(0)
4        matchSem = Semaphore(0)
```

Các biến boolean chỉ ra tài nguyên sẵn có hay không, pusher dùng *tobaccoSem* để đưa tín hiệu cho người hút có sẵn thuốc lá, các semaphore còn lại tương tự.

Dưới đây là code cho pusher lá thuốc trong số các pusher:

```
        tobacco.wait()
        mutex.wait()
        if isPaper:
            isPaper = False
            matchSem.signal()
        elif isMatch:
            isMatch = False
            paperSem.signal()
        else:
            isTobacco = True
        mutex.signal()
```

Người hút có sẵn lá thuốc:

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

## 2.2 Bài toán xe buýt Senate

### 2.2.1 Vấn đề

Bài toán được dựa trên việc đi xe buýt Sanate tại đại học Wellesley: những người cần đi xe sẽ tới trạm dừng và đợi xe buýt, khi xe tới tất cả những người đang đợi xe yêu cầu lên xe (*boardBus*), nhưng những người tới trạm lúc xe đã đang đón khách sẽ phải đợi xe sau. Xe có thể chở tối đa 50 người, nếu số người yêu cầu lên quá lớn thì một số sẽ phải đợi chuyến sau. Khi không còn khách nào đợi hoặc xe đã chở đủ người thì xe sẽ đi tiếp (*depart*), nếu se tới một trạm mà không có ai đang đợi thì nó sẽ lập tức rời trạm.

### 2.2.2 Giải pháp

Giải pháp sau được đề xuất bởi Grant Hutchins. Các biến được dùng :

```
    waiting = 0
    mutex = new Semaphore(1)
    bus = new Semaphore(0)
    boarded = new Semaphore(0)
```

*waiting* là số người đang đợi ở trạm, được bảo vệ bởi *mutex*. *bus* đưa ra tín hiệu khi có xe tới trạm, *boarded* đưa ra tín hiệu khi có thêm một người lên xe.

Code cho xe buýt :

```
mutex.wait()
n = min(waiting , 50)
for i in range(n):
    bus.signal()
    boarded.wait()

waiting = max(waiting -50, 0)
mutex.signal()

depart()
```

Code cho những người đang đợi:

```
mutex.wait()
waiting += 1
mutex.signal()

bus.wait()
board()
boarded.signal()
```

## 2.3 Bài toán ông già Noel

### 2.3.1 Vấn đề

Bài toán này là của William Stallings trong cuốn *Operating Systems*.
Ông già Noel ngủ tại cửa hàng của ông ấy tại cực Bắc và chỉ có thể được đánh thức bởi hoặc (1) tất cả 9 con tuần lộc trở về từ Nam Thái Bình Dương hoặc (2) 3 con yêu tinh gặp khó khăn trong việc chế tạo đồ chơi. Khi 3 yêu tinh đánh thức ông già Noel để giải quyết vấn đề của chúng thì những con yêu tinh còn lại phải đợi 3 con đang gặp trở về mới có thể lên gặp tiếp ông già Noel. Nếu ông già Noel tỉnh dậy để làm việc với 3 yêu tinh đang đợi mà trong lúc đó con tuần lộc cuối cùng vừa trở về, thì ông già Noel sẽ quyết định ưu tiên các con tuần lộc, các yêu tinh sẽ phải đợi đến sau Giáng sinh mới có thể tiếp tục gặp ông già Noel.

- Sau khi con tuần lộc thứ 9 trở về, ông già noel sẽ chuẩn bị xe kéo (*prepareSleigh*), sau đó các con tuần lộc se bắt đầu di chuyển (*getHitched*)
- Sau khi con yêu tinh thứ 3 tới, ông già Noel sẽ bắt đầu giúp chúng (*helpElves*), yêu tinh sẽ làm việc (*getHelp*)
- Cả 3 yêu tinh sẽ phải là việc luôn với ông già Noel trước khi yêu tinh khác yêu cầu

### 2.3.2 Giải pháp

Các biến khởi tạo:

```
elves = 0
reindeer = 0
santaSem = Semaphore(0)
reindeerSem = Semaphore(0)
elfTex = Semaphore(1)
mutex = Semaphore(1)
```

Cả elves và reindeer đều là biến đếm, được bảo vệ bởi mutex. Ông già Noel sẽ đợi cho đến khi yêu tinh hoặc tuần lộc gửi tín hiệu yêu cầu.
Dưới đây là code cho ông già Noel:

```
santaSem.wait()
mutex.wait()
if reindeer >= 9:
    prepareSleigh()
    reindeerSem.signal(9)
    reindeer -= 9
else if elves == 3:
    helpElves()
mutex.signal()
```

Khi ông già Noel thức dậy, ông sẽ kiểm tra số tuần lộc trở về có đủ 9, nếu không sẽ kiểm tra có đủ 3 yêu tinh hay chưa.
Nếu đủ 9 tuần lộc, ông già Noel sẽ *prepareSleigh*, sau đó gửi tín hiệu đến *reindeerSem* 9 lần để cho phép cả 9 con *getHitched*.
Nếu có yêu tinh đang đợi, ông già Noel sẽ yêu cầu *helpElves*, yêu tinh chỉ cần gửi tín hiệu tới *santaSem* để có thể được giúp ngay lập tức.

Code cho tuần lộc:

```
    mutex.wait()
    reindeer += 1
    if reindeer == 9:
        santaSem.signal()
    mutex.signal()

    reindeerSem.wait()
    getHitched()
```

Code cho yêu tinh:

```
    elfTex.wait()
    mutex.wait()
    elves += 1
    if elves == 3:
        santaSem.signal()
    else:
        elfTex.signal()
    mutex.signal()

    getHelp()

    mutex.wait()
    elves -= 1
    if elves == 0:
        elfTex.signal()
    mutex.signal()
```

## 2.4 Bài toán băng qua sông

### 2.4.1 Phát biểu bài toán

Bài toán vượt sông là một trong những lớp bài toán được viết bởi Anthony Joseph tại U.C.Berkeley. Bài toán này khá giống với bài toán $H_2O$, cụ thể là tồn tại một rào chắn chỉ cho phép một số lượng hữu hạn các luồng đi qua chỉ khi chúng được kết hợp theo một cách nào đấy.
Bài toán được phát biểu như sau: Ở một nơi gần thành phố Redmond, tiểu bang Washington D.C có một con thuyền được sử dụng bởi cả hacker Linux và nhân viên của Microsoft để vượt qua một con sông.Con thuyền chỉ chở được 4 người và nó sẽ không rời bến nếu có nhiều hoặc ít hơn 4 người trên thuyền. Để đảm bảo cho sự an toàn của khách hàng, thì ở trên thuyền không được có 1 hacker Linux và 3 nhân viên Microsoft hoặc 3 hacker Linux và 1 nhân viên Microsoft.
Khi mỗi luồng lên thuyền, nó sẽ gọi một hàm board(). Sau khi đủ 4 tiến trình lên thuyền, tức gọi hàm board() thì sẽ có duy nhất 1 tiến trình gọi hàm rowBoard().

### 2.4.2 Phân tích bài toán

Ở trên quan điểm về hệ điều hành, bài toán vượt sông chỉ ra rằng chiếc thuyền ở đây là tài nguyên găng. Tài nguyên găng chỉ được quyền sử dụng khi có đủ 4 luồng nhất định thỏa mãn điều kiện nào đó.
Biến hackers, serfs đếm số hacker Linux và số nhân viên Microsoft đang đợi ở thuyền. Bởi vì mỗi luồng đến yêu cầu tài nguyên găng tại thời điểm khác nhau, vì vậy chúng ta tạo khóa mutex để bảo vệ các luồng tránh kiểm tra điều kiện cùng một lúc.
Hàng đợi hackerQueue và serfQueue chỉ ra số lượng luồng đang đợi. Biến địa phương isCaptain quyết định xem luồng nào thực hiện hàm rowBoard().

```
1        barrier = Barrier(4)
2        mutex = Semaphore(1)
3        hackers = 0
4        serfs = 0
5        hackerQueue = Semaphore(0)
6        serfQueue = Semaphore(0)
7        local isCaptain = False
```

### 2.4.3 Giải pháp cho bài toán

Ý tưởng cơ bản cho bài toán là mỗi khi có một luồng đến, thì ta sẽ tăng bộ đếm và kiểm tra xem điều kiện đã thỏa mãn hay chưa. Phía dưới đây là code Python cho hacker Linux.

```
            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()  # captain keeps the mutex
        hackerQueue.wait()
        board()
        barrier.wait()
        if isCaptain:
            rowBoat()
            mutex.signal()  # captain releases the mutex
```

Đầu tiên, khi một luồng hacker tới đoạn găng, luồng sẽ kiểm tra xem khóa mutex có trạng thái như thế nào, nếu nó đang khóa thì chờ, nếu không thì hacker được tiến vào để kiểm tra điều kiện đủ thành viên hay không. Điều kiện thỏa mãn là có 4 hacker hoặc 2 hacker 2 nhân viên. Khi đã đủ điều kiện, lập tức đánh thức các luồng khác đang ngủ và chọn ra Captain để thực thi đoạn găng. Nếu không, thì mở khóa mutex, đưa luồng vào hàng đợi để ngủ.
Ngoài ra tiếp theo có các hàm phía sau để khi các luồng tiếp theo đến đánh thức luồng đang ngủ. Trước khi thực thi, kiểm tra xem luồng nào được cấp quyền điều khiển chính và thực thi, tại đó barrier là khóa để đoạn găng được thực thi và chỉ Captain mới có quyền rowboat() và thả khóa mutex.

### 2.4.4 Các vấn đề còn tồn tại

- Giải pháp trên khá hữu hiệu, tuy nhiên còn một số vấn đề về nạn đói. Vấn đề cần quan tâm là việc luồng nào sẽ được đánh thức để lên thuyền. Nếu lựa chọn không hợp lý, sẽ làm cho một số luồng không bao giờ được lựa chọn.

- Chúng ta có thể đặt độ ưu tiên cho các luồng, các luồng ngủ càng lâu thì độ ưu tiên càng tăng

## 2.5 Bài toán về phòng cho bữa tiệc

### 2.5.1 Phát biểu bài toán

Bài toán được phát biểu dựa trên sinh viên và nhà quản lý sinh viên. Bài toán được phát biểu như sau.

- Bất cứ học sinh nào cũng có thể ở cùng nhau trong cùng một thời điểm.
- Nhà quản lý sinh viên chỉ có thể vào phòng khi mà trong phòng không còn một sinh viên nào (để kiểm tra) hoặc nếu có trên 50 sinh viên ở trong phòng (để phá vỡ bữa tiệc).
- Trong khi nhà quản lý ở trong phòng thì không ai có thể vào phòng, tuy nhiên thì sinh viên có thể ra khỏi phòng.
- Nhà quản lý sinh viên có thể không rời phòng cho tới khi tất cả các sinh viên rời khỏi phòng.
- Chỉ có duy nhất một nhà quản lý sinh viên.

### 2.5.2 Phân tích bài toán

Ở góc nhìn về hệ điều hành, bài toán nói về vấn đề tài nguyên găng trên góc độ luồng, tiến trình. Có thể giữa nhà quản lý và sinh viên thì phòng KTX là tài nguyên găng, tuy nhiên thì giữa các sinh viên thì KTX không phải là tài nguyên găng. Vì vậy chúng ta chỉ cần đồng bộ cho sinh viên và nhà quản lý.

```
1         students = 0
2       dean = "waiting"
3       mutex = Semaphore(1)
4       turn = Semaphore(1)
5       clear = Semaphore(0)
6       lieIn = Semaphore(0)
```

Biến students biểu thị số lượng sinh viên trong phòng, biến dean biểu thị trạng thái của nhà quản lý "waiting" hay "in room". Khóa mutex, turn để giữ cho sinh viên không thể vào khi có nhà quản lý sinh viên ở trong phòng.

### 2.5.3 Giải pháp cho bài toán

Theo như tác giả của bài toán này, đây là một bài toán thực sự khó. Có thêm một phiên bản khác cho bài toán, đó là nhà quản lý sinh viên có thể vào phòng bất cứ lúc nào. Tuy nhiên, nếu trong phòng không tổ chức tiệc, nhà quản lý sẽ ra về với bộ mặt xấu hổ.
Matt Tesch đã viết ra một lời giải để tránh sự xấu hổ của nhà quản lý, tuy nhiên kết quả khá phức tạp. Sau đây là đoạn mã dễ đọc hơn nhiều.

```
        mutex.wait()
        if students > 0 and students < 50:
            dean = "waiting"
            mutex.signal()
            lieIn.wait()          # and get mutex from the student .

        # students must be 0 or >= 50

        if students >= 50:
            dean = "in the room"
            breakup()
            turn.wait()           # lock the turnstile
            mutex.signal()
            clear.wait()          # and get mutex from the student .
            turn.signal()         # unlock the turnstile
        else :
            search()              # students must be 0

        dean = "not here"
        mutex.signal()
```

Khi Nhà quản lý đến, có 3 trường hợp có thể xảy ra:

- Nếu có sinh viên trong phòng, nhưng số lượng lại nhỏ hơn 50, Nhà quản lý phải đợi.
- Nếu có nhiều hơn 50 sinh viên, Nhà quản lý phá vỡ bữa tiệc và đợi cho tất cả sinh viên rời phòng.
- Nếu không có sinh viên nào, Nhà quản lý được quyền vào và rời phòng một cách bình thường.

Ở trong hai trường hợp đầu, Nhà quản lý phải đợi điều kiện xảy ra, vì vậy nhà quản lý phải thả khóa mutex, nếu không thì sinh viên sẽ không được vào phòng. Điều đó gây nên bế tắc. Khi luồng Nhà quản lý được đánh thức, Nhà quản lý phải chấm điểm cho sinh viên, vì vậy Nhà quản lý phải lấy khóa mutex.

Algorithm 1: Code for Student

```
            mutex . wait ()
        if dean == "in the room":
            mutex.signal()
            turn.wait()
            turn.signal()
            mutex.wait()

        students += 1

        if students == 50 and dean == "waiting":
            lieIn.signal()
        # and pass mutex to the dean
```

```
13          else :
14              mutex.signal()
15
16  party()
17
18          mutex.wait()
19          students -= 1
20
21          if students == 0 and dean == "waiting":
22              lieIn.signal()      # and pass mutex to the dean
23          elif students == 0 and dean == "in the room":
24              clear.signal()      # and pass mutex to the dean
25          else :
26              mutex.signal()
```

Khi có tín hiệu từ nhà quản lý, có 3 trường hợp có thể xảy ra:

- Nếu Nhà quản lý đang chờ, thì sinh viên thứ 50 hoặc sinh viên ra khỏi phòng cuối cùng phải thả khóa lieIn.
- Nếu nhà quản lý đang ở trong phòng(chờ sinh viên ra khỏi phòng), thì sinh viên ra khỏi phòng cuối cùng phải thả khóa clear.
- Trong cả 3 trường hợp, phải hiểu rằng khóa đều được truyền từ sinh viên sang nhà quản lý.

## 2.6 Bài toán bể bơi

### 2.6.1 Phát biểu bài toán

Bài toán bể bơi là một bài toán đồng bộ giữa 2 sự kiện đến và rời đi của các người đi bơi tại một cơ sở hồ bơi. Có hai lớp tài nguyên găng, và cả hai đều bị hạn chế khả năng cung cấp là $n$ phòng thay đồ và $k$ giỏ đồ, thường thường $n < k$. Các hoạt động của tiến trình:

- Tìm giỏ đồ và phòng thay đồ có sẵn.
- Mặc đồ bơi và để quần áo vào giỏ đồ.
- Rời khỏi phòng thay đồ và gửi lại giỏ đồ cho nhân viên quản lý.
- Bơi. Giả sử rằng bể bơi là tài nguyên không bị giới hạn về khả năng phục vụ.
- Lấy lại giỏ đồ từ nhân viên quản lý.
- Tìm một phòng thay đồ trống và thay đồ. Bài toán này khá giống với bài toán nhà triết gia ăn tối. Bế tắc có thể xảy ra khi mà có một người đến chờ lấy giỏ đồ, và có người cầm giỏ đồ chờ phòng thay đồ.

# 3 Tổng kết

Lớp các bài toán đồng hộ hóa tiến trình luôn là một vấn đề nan giải cho các nhà lập trình đa luồng, bởi tính phức tạp và rất dễ gây lỗi. Khi lập trình đa luồng luôn cần phải đảm bảo đoạn găng cần được truy cập đúng như điều kiện đã cho, tránh xảy ra các nạn đói, bế tắc gây crash chương trình. Trên đây là phần trình bày về một số bài toán về đồng bộ tiến trình, một số lời giải còn chưa tối ưu, tuy nhiên cũng đã giải quyết một phần bài toán đặt ra.

# 4 Reference

1. Downey, A. (2008). The little book of semaphores. Green Tea Press.
2. Berztiss, A. T. (1982). Synchronization of processes.