1 Bài toán người hút thuốc

1.1 Phát biểu bài toán

- Có một cửa hàng và 3 người hút thuốc lá
- Một người cần có đủ 3 nguyên liệu sau để có thể hút thuốc: thuốc lá, giấy và diêm
- Cửa hàng có thể cung cấp một số lượng không giới hạn các nguyên liệu này.
- Mỗi một người trong 3 người hút thuốc có một số lượng không giới hạn một trong 3 nguyên liệu trên, nghĩa là một người có thuốc lá, một người có giấy và một người có diêm.
- Cửa hàng thực hiện lặp lại công việc sau: chọn ngẫu nhiên ra 2 loại nguyên liệu khác nhau và bán nó cho người hút thuốc cần 2 loại nguyên liệu này (ví dụ nếu cửa hàng chọn diêm và giấy thì cửa hàng sẽ bán nó cho người có thuốc lá).

1.2 Ý nghĩa của bài toán

Bài toán người hút thuốc là một trong những bài toán kinh điển về vấn đề đồng bộ hóa tiến trình. Nó là sự mô hình hóa của một vấn đề thực tế đó là việc phân phối tài nguyên cần thiết cho các tiến trình. Trong thực tế, việc phân phối tài nguyên cho các tiến trình có thể xảy ra tình huống sau đây:

- Có một số tiến trình đang chờ hệ thống cấp phát tài nguyên.
- Nếu lượng tài nguyên sẵn có trong hệ thống là có thể cung cấp được cho một số tiến trình, hệ thống cần đánh thức các tiến trình này để chúng tiếp tục thực hiện. Ngược lại, đối với các tiến trình không thể hoạt động với lượng tài nguyên sẵn có, cần phải tránh việc đánh thức chúng.
- Bài toán đặt ra là phải lựa chọn các tiến trình phù hợp để đánh thức.

Đây chính là tiền đề dẫn đến bài toán người hút thuốc. Có thể thấy rõ sự tương ứng giữa vấn đề trong thực tế với bài toán này:

- Cửa hàng đại diện cho hệ điều hành, tức thành phần phân phối tài nguyên.
- 3 người hút thuốc đại diện cho các tiến trình trong hệ thống yêu cầu các tài nguyên khác nhau.

1.3 Các phiên bản của bài toán

1. Phiên bản "không thể" (impossible version): Suhas Patil là người đã đặt ra các ràng buộc cho phiên bản này, đó là:
   - Thứ nhất, không được thay đổi mã nguồn của tiến trình cửa hàng.
   - Thứ hai, không được sử dụng các câu lệnh điều kiện hay một mảng các đèn báo.

   Với 2 ràng buộc này, rất nhiều vấn đề trở nên không thể giải quyết được.
2. Phiên bản "thú vị" (interesting version): Tương tự như phiên bản phía trên, nhưng không có ràng buộc thứ hai.
3. Phiên bản tầm thường (trivial version): Trong một số sách, bài toán quy định rằng cửa hàng sẽ biết trước về nhu cầu của những người hút thuốc và chọn ra người phù hợp để cung cấp nguyên liệu. Rõ ràng điều này là không thực tế trong cả trường hợp của một người bán hàng thực sự và trường hợp của hệ điều hành. Nó khiến cho việc các tiến trình đòi hỏi tài nguyên khác nhau không còn ý nghĩa bởi hệ điều hành luôn dựa vào yêu cầu của tiến trình để cung cấp chính xác những gì mà tiến trình đòi hỏi. Đồng thời, đối với phiên bản này, bài toán trở nên quá dễ để giải quyết, đó là lý do phiên bản này có tên là phiên bản tầm thường, hay phiên bản không thú vị.

1.4 Giải quyết bài toán

Dựa trên những nhận xét trên, chúng ta sẽ chỉ tập trung xem xét phiên bản "thú vị" của bài toán người hút thuốc. Để thỏa mãn ràng buộc của bài toán, chúng ta cần phải xác định mã nguồn của tiến trình cửa hàng. Cửa hàng sẽ sử dụng các đèn báo sau đây:

Agent semaphores

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

Tiến trình cửa hàng thực chất bao gồm 3 tiến trình:

- Tiến trình A sản xuất thuốc lá và giấy.
- Tiến trình B sản xuất giấy và diêm.
- Tiến trình C sản xuất thuốc lá và diêm.

Mỗi khi tiến trình cửa hàng hoạt động, chỉ có một trong 3 tiến trình A, B, C được phép hoạt động. Do đó, chúng ta sử dụng đèn báo `agentSem` để điều phối hoạt động của 3 tiến trình này. Mỗi khi đèn báo `agentSem` được `signal`, một trong 3 tiến trình A, B, C sẽ được đánh thức và sản xuất 2 loại nguyên liệu tương ứng bằng cách `signal` 2 trong số 3 đèn báo: `tobacco, paper, matches` tương ứng với tiến trình đó.

Cửa hàng A

```
do{
   agentSem.wait()
   tobacco.signal()
   paper.signal()
}while(1)
```

Cửa hàng B

```
do{
   agentSem.wait()
   paper.signal()
   match.signal()
}while(1)
```

Cửa hàng C

```
do{
   agentSem.wait()
   paper.signal()
   match.signal()
}while(1)
```

Đối với các tiến trình người hút thuốc, công việc cần thực hiện của các tiến trình này là:

- Chờ nguyên liệu thứ nhất.
- Chờ nguyên liệu thứ hai
- Hút thuốc

Ở đây, chúng ta có 2 khả năng cần được xem xét:

- Thứ nhất, cửa hàng chờ tiến trình người hút thuốc nhận được nguyên liệu mới tiếp tục sản xuất nguyên liệu.
- Thứ hai, cửa hàng luôn sản xuất nguyên liệu mà không cần quan tâm đến việc người hút thuốc đã nhận được nguyên liệu hay chưa.

1.4.1 Trường hợp 1: Cửa hàng chờ người hút thuốc

Trong trường hợp này, tiến trình người hút thuốc cần làm thêm một công việc đó là đánh thức tiến trình cửa hàng mỗi khi nó thực hiện xong. Mã nguồn của tiến trình người hút thuốc chỉ đơn giản như sau:

Người có diêm

```
do{
   tobacco.wait()
   paper.wait()
   agentSem.signal()
}while(1)
```

Người có thuốc

```
do{
   paper.wait()
   match.wait()
   agentSem.signal()
}while(1)
```

Người có giấy

```
do{
   tobacco.wait()
   match.wait()
   agentSem.signal()
}while(1)
```

Chúng ta có thể thấy rằng, giải pháp này có khả năng gây ra bế tắc. Giả sử tiến trình **Cửa hàng A** được gọi đầu tiên, nó sẽ sản xuất ra thuốc và giấy. Lúc này, cả hai tiến trình **người có thuốc** và **người có giấy** cùng hoạt động. Tiến trình **người có thuốc** lấy được giấy và tiến trình **người có giấy** lấy được thuốc. Nhưng sau đó, cả 2 tiến trình này cùng bị chặn do không lấy được diêm. Trong khi đó, tiến trình **người có diêm** cũng bị chặn vì thuốc và giấy đã bị 2 tiến trình kia chiếm mất. Do cả 3 tiến trình người hút thuốc đều không thể kết thúc, đèn báo `agentSem` không được giải phóng và tiến trình cửa hàng cũng bị chặn, hệ thống rơi vào trạng thái bế tắc.

Giải pháp đưa ra ở đây là sử dụng thêm 3 tiến trình hỗ trợ. Tưởng tượng rằng mỗi khi cửa hàng sản xuất ra nguyên liệu, nó đem đặt các nguyên liệu sản xuất lên một chiếc bàn. Mỗi một tiến trình hỗ trợ sẽ thực hiện công việc đặt một loại sản phẩm nhất định lên trên bàn. Giả sử tiến trình **Pusher A** thực hiện công việc đặt thuốc lá lên bàn. Nếu nó thấy trên bàn đã có diêm hoặc giấy, nó sẽ lấy một trong hai nguyên liệu này kết hợp với thuốc lá mà nó mang đến để đưa cho người phù hợp. Ngược lại, nó sẽ chỉ đặt thuốc lên bàn. Tương tự, các tiến trình **Pusher B** và **Pusher C** làm các công việc đặt diêm và giấy lên bàn. Để thực hiện lời giải pháp này, chúng ta cần sử dụng thêm một số biến và đèn báo sau:

```
isTobacco = isPaper = isMatch = false
tobaccoSem = Semaphore(0)
paperSem = Semaphore(0)
matchSem = Semaphore(0)
mutex = Semaphore(1)
```

Các biến logic `isTobacco, isPaper, isMatch` dùng để xác định xem các nguyên liệu tương ứng đã có ở trên bàn hay chưa. Các đèn báo `tobaccoSem, paperSem, matchSem` dùng để đánh thức các tiến trình người hút thuốc tương ứng. Bây giờ, các tiến trình người hút thuốc không chờ 2 nguyên liệu còn thiếu mà chờ tín hiệu từ đèn báo tương ứng. Cũng cần lưu ý rằng, tại mỗi thời điểm, chỉ một trong 3 tiến trình hỗ trợ được phép hoạt động nếu không sẽ dẫn đến mất kiểm soát, do đó đèn báo `mutex` được sử dụng. Mã nguồn của 3 tiến trình hỗ trợ và 3 tiến trình người hút thuốc được viết như sau:

Pusher A

```
do{
   tobacco.wait()
   mutex.wait()
      if (isPaper){
         isPaper = false
         matchSem.signal()
      }
      else if (isMatch){
         isMatch = false
         paperSem.signal()
      }
      else
         isTobacco = true
   mutex.signal()
}while(1)
```

Pusher B

```
do{
   match.wait()
   mutex.wait()
      if (isPaper){
         isPaper = false
         tobaccoSem.signal()
      }
      else if (isTobacco){
         isTobacco = false
         paperSem.signal()
      }
      else
         isMatch = true
   mutex.signal()
}while(1)
```

Pusher C

```
do{
   paper.wait()
   mutex.wait()
      if (isTobacco){
         isTobacco = false
         matchSem.signal()
      }
      else if (isMatch){
         isMatch = false
         tobaccoSem.signal()
      }
      else
         isPaper = true
   mutex.signal()
}while(1)
```

Người có diêm

```
do{
   matchSem.wait()
   makeCigarette()
   agentSem.signal()
   smoke()
}while(1)
```

Người có thuốc

```
do{
   tobaccoSem.wait()
   makeCigarette()
   agentSem.signal()
   smoke()
}while(1)
```

Người có giấy

```
do{
   paperSem.wait()
   makeCigarette()
   agentSem.signal()
   smoke()
}while(1)
```

Như vậy, với việc thêm vào 3 tiến trình hỗ trợ, chúng ta đã giải quyết được bài toán người hút thuốc.

1.4.2 Trường hợp 2: Cửa hàng không chờ người hút thuốc

Đây là trường hợp tổng quát hơn của trường hợp 1. Ở đây, thay vì sử dụng các biến logic `isTobacco, isPaper, isMatch`, chúng ta sẽ sử dụng các biến kiểu nguyên `numTobacco, numPaper, numMatch`. Các biến này dùng để chỉ ra số lượng mỗi loại nguyên liệu có trên bàn. Ở đây, thay vì tưởng tượng các tiến trình hỗ trợ đặt nguyên liệu lên bàn, ta có thể tưởng tượng rằng các tiến trình đang ghi điểm vào một bảng điểm có 3 cột. Trong trường hợp này, các tiến trình hỗ trợ có hành vi giống hệt ở trườn hợp 1, chỉ khác là ở trong bảng điểm, số lượng các loại nguyên liệu có thể lớn hơn 1. Mỗi khi cửa hàng sản xuất xong, đèn báo `agentSem` sẽ được giải phóng ngay chứ không phải đợi một trong số các tiến trình người hút thuốc giải phóng. Mã nguồn của toàn bộ bài toán được tổng hợp lại như sau:

Cửa hàng A

```
do{
   agentSem.wait()
   tobacco.signal()
   paper.signal()
}while(1)
```

Cửa hàng B

```
do{
   agentSem.wait()
   paper.signal()
   match.signal()
}while(1)
```

Cửa hàng C

```
do{
   agentSem.wait()
   paper.signal()
   match.signal()
}while(1)
```

Pusher A

```
do{
   tobacco.wait()
   mutex.wait()
      if (numPaper > 0){
         numPaper--
         matchSem.signal()
      }
      else if (numMatch > 0){
         numMatch--
         paperSem.signal()
      }
      else
         numTobacco++
   mutex.signal()
}while(1)
```

Pusher B

```
do{
   match.wait()
   mutex.wait()
      if (numPaper > 0){
         numPaper--
         tobaccoSem.signal()
      }
      else if (numTobacco > 0){
         numTobacco--
         paperSem.signal()
      }
      else
         numMatch++
   mutex.signal()
}while(1)
```

Pusher C

```
do{
   paper.wait()
   mutex.wait()
      if (numTobacco > 0){
         numTobacco--
         matchSem.signal()
      }
      else if (numMatch > 0){
         numMatch--
         tobaccoSem.signal()
      }
      else
         numPaper++
   mutex.signal()
}while(1)
```

Người có diêm

```
do{
   matchSem.wait()
   makeCigarette()
   agentSem.signal()
   smoke()
}while(1)
```

Người có thuốc

```
do{
   tobaccoSem.wait()
   makeCigarette()
   agentSem.signal()
   smoke()
}while(1)
```

Người có giấy

```
do{
   paperSem.wait()
   makeCigarette()
   agentSem.signal()
   smoke()
}while(1)
```

2 Bài toán ông già Noel

2.1 Phát biểu bài toán

- Có một ông già Noel, 9 chú tuần lộc và vô số yêu tinh.
- Giả sử ban đầu, ông già Noel đang ngủ tại cửa hàng đồ chơi ở Bắc Cực, 9 chú tuần lộc đang ở vùng nhiệt đới phía Nam, các yêu tinh đang làm đồ chơi tại cửa hàng.
- Ông già Noel chỉ được đánh thức khi có một trong 2 điều kiện
  - Có 3 yêu tinh gặp khó khăn trong khi làm đồ chơi và cần ông già Noel giúp đỡ.
  - Cả 9 chú tuần lộc đều đã trở về Bắc cực.
- Nếu cả 9 chú tuần lộc đều trở về và đồng thời có 3 yêu tinh cần giúp đỡ thì ông già Noel ưu tiên việc chuẩn bị xe trượt tuyết hơn; các yêu tinh phải đợi đến khi xe tuyết được chuẩn bị xong.
- Khi có 3 yêu tinh đợi sự giúp đỡ của ông già Noel, các yêu tinh khác cần đến sự giúp đỡ của ông già Noel phải chờ cho đến khi 3 yêu tinh kia trở về.
- Khi chú tuần lộc thứ 9 chưa đến, các chú tuần lộc khác phải đợi mà không được đánh thức ông già Noel.
- Hoạt động của ông già Noel được đặt trong một vòng lặp vô hạn để giải quyết được nhiều vấn đề nhất có thể. Trong khi đó, các chú tuần lộc không muốn trở về Bắc cực và chỉ trở về khi không còn lựa chọn nào khác, còn các yêu tinh không phải luôn luôn cần đến sự giúp đỡ của ông già Noel.

2.2 Ý nghĩa của bài toán

Bài toán ông già Noel thuộc lớp bài toán ít kinh điển hơn về đồng bộ tiến trình. Bài toán này thuộc kiểu hình mẫu bài toán barrier, trong đó một tổ hợp các tiến trình được phép đi qua barrier có độ ưu tiên cao hơn một tổ hợp khác. Đây là vấn đề xảy ra trong thực tế, ví dụ:

- Trong hệ thống có một số tiến trình chỉ được phép hoạt động khi có đồng thời một số lượng nhất định các tiến trình cùng loại.
- Khi số lượng tiến trình cần thiết để hoạt động chưa đủ, các tiến trình đến trước phải chờ.
- Nếu tất cả các tiến trình đều đang chờ, hệ điều hành tạm thời không hoạt động (thực tế hệ điều hành luôn luôn hoạt động, ta chỉ giả sử trong hệ thống lúc này chỉ có các tiến trình đang chờ).
- Các tiến trình thuộc các kiểu khác nhau có độ ưu tiên khác nhau và hệ điều hành sẽ phục vụ các tiến trình có độ ưu tiên cao hơn trước.

Vấn đề nêu trên được ánh xạ qua bài toán ông già Noel với sự tương ứng như sau:

- Ông già Noel đại diện cho hệ điều hành.
- Yêu tinh đại diện cho kiểu tiến trình có độ ưu tiên thấp.
- Tuần lộc đại diện cho kiểu tiến trình có độ ưu tiên cao.
- Số lượng tiến trình loại tuần lộc được phép hoạt động là 9 và số lượng tiến trình loại yêu tinh được phép hoạt động là 3.

2.3 Giải quyết bài toán

Để giải quyết bài toán, ta sẽ sử dụng các biến và đèn báo sau đây:

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

- `elves` và `reindeer` là các biến đếm để đếm số lượng yêu tinh và tuần lộc đang chờ ông già Noel. Cả 2 biến đếm này đều được bảo về bởi đèn báo `mutex`.
- Ông già Noel đợi ở đèn báo `santaSem` trước khi được đánh thức bởi 9 chú tuần lộc hoặc 3 yêu tinh.
- Các chú tuần lộc đến trước đợi ở đèn báo `reindeer` và chỉ được đánh thức bởi ông già Noel khi chú tuần lộc thứ 9 đánh thức ông già Noel.
- Đèn báo `elfTex` được sử dụng để ngăn chặn các yêu tinh khác khi có 3 yêu tinh đang được ông già Noel giúp đỡ.

2.3.1 Tiến trình ông già Noel

Ông già Noel chỉ phải thực hiện các công việc sau:

- Đợi cho đến khi được đánh thức.
- Nếu được đánh thức, kiểm tra xem số lượng tuần lộc đã đủ 9 hay chưa. Nếu đủ, ông già Noel sẽ đi chuẩn bị xe trượt tuyết. Ngược lại, ông già Noel sẽ giúp đỡ 3 yêu tinh.
- Lặp lại 2 công việc trên.

Mã nguồn của tiến trình ông già Noel được viết như sau:

Santa Claus's code

```
do{
   santaSem.wait()
   mutex.wait()
      if (reindeer == 9){
         prepareSleigh()
         reindeerSem.signal(9)
         reindeer -= 9
      else
         helpElves()
   mutex.signal()
}while(1)
```

2.3.2 Tiến trình tuần lộc

Tiến trình tuần lộc là tiến trình đơn giản nhất trong số 3 tiến trình: ông già Noel, tuần lộc và yêu tinh. Công việc của tuần lộc chỉ bao gồm:

- Đợi ở đèn báo `reindeerSem`.
- Nếu chú tuần lộc thứ 9 đến, nó đánh thức ông già Noel. Sau đó, ông già Noel đánh thức cả 9 chú tuần lộc và chúng được buộc vào xe trượt tuyết.

Mã nguồn của tiến trình tuần lộc được viết như sau:

Reindeer's code

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

2.3.3 Tiến trình yêu tinh

Tiến trình yêu tinh hoạt động tương tự như tiến trình tuần lộc ngoại trừ ràng buộc: khi có 3 yêu tinh đang được ông già Noel giúp đỡ, các yêu tinh khác cần giúp đỡ phải chờ cho đến khi 3 yêu tinh kia trở về.
Mã nguồn:

Elf's code

```
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 yêu tinh đầu tiên đến sẽ giải phóng đèn báo `elfTex` ngay trước khi giải phóng đèn báo `mutex`. Nhưng yêu tinh thứ 3 sẽ không giải phóng đèn báo `elfTex` cho đến khi cả 3 yêu tinh đều đã được giúp đỡ. Điều này ngăn chặn các yêu tinh khác xin giúp đỡ trong khi 3 yêu tinh này đang được giúp đỡ.
- Yêu tinh cuối cùng rời khỏi sẽ giải phóng đèn báo `elfTex` và cho phép các yêu tinh khác được phép xin giúp đỡ từ ông già Noel.

3 Bài toán qua sông

3.1 Phát biểu bài toán

- Ở một vùng gần Redmon, Washington, có một con thuyền được sử dụng bởi cả các hackers của Linux và lập trình viên của Microsoft để qua sông.
- Con thuyền chỉ chở được chính xác 4 người. Thuyền sẽ không thể đi nếu có nhiều hơn hay ít hơn 4 người.
- Để đảm bảo sự an toàn của hành khách, thuyền không cho phép chở 1 hacker của Linux với 3 lập trình viên của Microsoft, cũng như 1 lập trình viên của Microsoft với 3 hackers của Linux. Tất cả các tổ hợp còn lại đều được phép.
- Cần đảm bảo cả 4 người trong một tổ hợp thỏa mãn phải lên thuyền trước khi có bất kì người nào khác lên thuyền. Sau đó, một trong số 4 người sẽ là thuyền trưởng và lái con thuyền qua sông.

3.2 Ý nghĩa của bài toán

Bài toán người qua sông thuộc lớp bài toán ít kinh điển hơn về đồng bộ hóa tiến trình. Bài toán này là một trường hợp cụ thể của các bài toán tổng quát về barrier và queue. Yêu cầu của bài toán là chỉ cho phép các tổ hợp tiến trình phù hợp được phép đi qua. Ràng buộc này khá giống với bài toán ông già Noel nhưng khác nhau ở chỗ tổ hợp các tiến trình trong bài toán này có thể bao gồm các tiến trình thuộc nhiều loại khác nhau. Bài toán này miêu tả vấn đề thực tế sau đây:

- Trong hệ thống có một số lượng các tiến trình thuộc nhiều loại khác nhau đang chờ được thực thi.
- Giả sử lúc đầu không tồn tại một tổ hợp tiến trình nào thỏa mãn để có thể được thực thi, các tiến trình phải xếp hàng trong một hàng đợi tương ứng với kiểu của mình.
- Trong khi các tiến trình đang chờ đợi, các tiến trình mới có thể xuất hiện và yêu cầu được thực thi. Các tiến trình mới xuất hiện này sẽ tạo nên các tổ hợp thỏa mãn.
- Mỗi khi một tiến trình mới xuất hiện, nó kiểm tra xem liệu nó cùng với các tiến trình hiện đang ở trong hàng đợi có tạo thành một tổ hợp thỏa mãn hay không.
- Nếu tìm được một tổ hợp, tiến trình mới đến sẽ đánh thức tất cả các tiến trình trong tổ hợp thỏa mãn này, cho phép tất cả được thực thi và chặn tất cả các tiến trình khác trong khi tổ hợp này chưa hoạt động xong.
- Ngược lại, nếu không tìm được tổ hợp nào thỏa mãn, tiến trình mới đến xếp vào cuối hàng đợi tương ứng với kiểu của mình.

Bài toán thực tế trên đây được phát biểu bằng một cách dễ hiểu và trực quan hơn thông qua bài toán qua sông, trong đó:

- Hackers của Linux và lập trình viên của Microsoft đại diện cho 2 loại tiến trình đang có mặt trong hệ thống.
- Con thuyền qua sông biểu diễn cho hoạt động của một tổ hợp tiến trình được phép hoạt động.
- Các hàng đợi tiến trình tương ứng với hàng hackers của Linux và lập trình viên của Microsoft đợi lên thuyền.

3.3 Giải quyết bài toán

Chúng ta cần tìm cách để đáp ứng được các ràng buộc của bài toán. Trong giải pháp đưa ra dưới đây, chúng ta sẽ sử dụng các biến và đèn báo sau:

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

- `barrier` là một rào chắn đảm bảo cho các hành khách đi qua chỉ khi số lượng đạt đủ 4.
- Đèn báo `mutex` dùng để bảo vệ các biến `hackers` và `serfs`.
- Biến đếm `hackers` và `serfs` dùng để đếm số lượng các hackers và lập trình viên.
- Nếu một hành khách đến mà chưa tìm thấy tổ hợp nào phù hợp để có thể qua sông, hành khách này phải đợi ở hàng đợi tương ứng với kiểu của mình. 2 đèn báo `hackerQueue` và `serfQueue` được sử dụng để quản lý 2 hàng đợi hacker và lập trình viên tương ứng.
- Mỗi hành khách đến sẽ có một trạng thái cục bộ `isCaptain`. Biến logic này chỉ ra liệu hành khách này có phải là thuyền trưởng của con thuyền hay không (thuyền trưởng là người đến cuối cùng, đánh thức 3 người còn lại và lái thuyền qua sông).

Các hành khách thuộc kiểu hacker và kiểu lập trình viên có vai trò tương đương nhau trong bài toán này. Do đó, mã của hai kiểu tiến trình này là hoàn toàn đối xứng. Dưới đây là mã nguồn của tiến trình kiểu hacker Linux:

Hacker Linux's code

```
mutex.wait()
  hackers += 1
  if (hackers == 4){
    hackerQueue.signal(4)
    hackers = 0
    isCaptain = true
  }
  else if (hackers == 2 && serfs ≥ 2){
    hackerQueue.signal(2)
    serfQueue.signal(2)
    hackers = 0
    serfs -= 2
    isCaptain = true
  }
  else
    mutex.signal()

hackerQueue.wait()

board()
barrier.wait()

if (isCaptain){
  rowBoat()
  mutex.signal()
```

- Đầu tiên, khi một tiến trình **Hacker Linux** đến, nó cần phải kiểm tra xem đèn báo `mutex` có đang tự do hay không. Nếu có, tiến trình chiếm đèn báo `mutex`, ngược lại, nó bị chặn tại vị trí này.
- Sau khi chiếm đèn báo `mutex`, tiến trình tăng biến đếm `hackers` lên 1.
- Sau đó, tiến trình kiểm tra xem có tổ hợp nào phù hợp để có thể lên thuyền hay không. Nếu có, nó đánh thức 3 tiến trình phù hợp và gán cho mình chức thuyền trưởng. Ngược lại, nó sẽ giải phóng đèn báo `mutex`.

- Nếu tiến trình chưa tìm được tổ hợp nào phù hợp, bước tiếp theo nó sẽ phải đợi ở đèn báo `hackerQueue`.
- Sau khi có một tiến trình phù hợp đánh thức 3 tiến trình khác để tạo thành một tổ hợp thỏa mãn, cả 4 tiến trình cùng đi lên thuyền. Tiến trình nào lên trước sẽ phải đợi tại rào chắn `barrier` cho đến khi cả 4 tiến trình đều đã lên thuyền.
- Sau khi cả 4 tiến trình đều đã lên thuyền, tiến trình thuyền trưởng sẽ lái thuyền sang sông và sau đó giải phóng đèn báo `mutex`. Trước khi con thuyền qua sông, đèn báo `mutex` ngăn chặn tất cả các tiến trình khác khỏi việc lên thuyền.