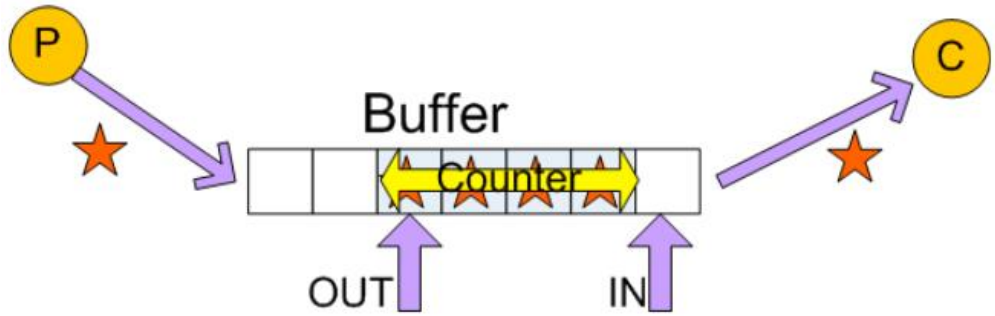
- Người sản xuất-người tiêu thụ (Producer-Consumer)
- Bữa ăn tối của triết gia (Dining Philosophers)
- Người đọc và biên tập viên (Readers-Writers)
- Người thợ cắt tóc ngủ gật (Sleeping Barber)
- Bathroom Problem
- Đồng bộ theo Barriers

Bài toán người sản xuất (producer)-người tiêu thụ(consumer)




```
while(1) {
    /*produce an item in  Buffer*/
     while (Counter == BUFFER_SIZE) ;
                /*do nothing*/
    Buffer[IN] = nextProduced;
    IN = (IN + 1) % BUFFER_SIZE;
    Counter++;
}
```

Producer

```
while(1){
    while(Counter == 0) ;
            /*do nothing*/
     nextConsumed = Buffer[OUT];
     OUT =(OUT + 1) % BUFFER_SIZE;
    Counter--; /*consume the item in
                    nextConsumed*/
}
```

Consumer

Bài toán người sản xuất (producer)-người tiêu thụ(consumer)

Giải pháp: Dùng 1 đèn báo Mutex để diều độ biến Counter

Khởi tạo: Mutex = 1

```
while(1) {
      /*produce a product in  Buffer */
     if(Counter==SIZE) block();
                    /*do nothing*/
      Buffer[IN] = nextProduced;
      IN = (IN + 1) % BUFFER_SIZE;
      Counter++;
if(Counter==1) wakeup(Consumer);
}
```

Producer

```
while(1){
      if(Counter == 0) block();
              /*do nothing*/
       nextConsumed = Buffer[OUT];
       OUT =(OUT + 1) % BUFFER_SIZE;
      Counter--;
if(Counter==SIZE-1) wakeup(Producer);
/*consume the item in   Buffer*/
}
```

Consumer

Vấn đề: Giả thiết Counter=0

- Consumer kiểm tra counter => gọi thực hiện lệnh block()
- Producer tăng counter lên 1 và gọi wakeup(Consumer)
- Consumer chưa bị block => Câu lệnh wakeup() bị bỏ qua

Bài toán người sản xuất (producer)-người tiêu thụ(consumer)

Giải pháp 2: Dùng 2 đèn báo full, empty được khởi tạo:
full ← 0 : Số phần tử trong buffer
empty←BUFFER_SIZE: Số chỗ trống trong buffer

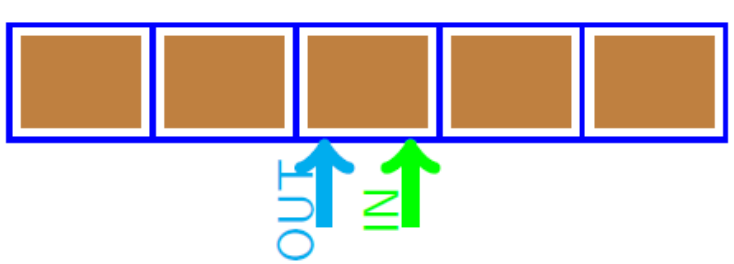
```
do{
{Tạo phần tử mới}
wait(empty);
{Đặt phần tử mới vào Buffer}
signal(full);
} while (1);
```

Producer

```
do{
wait(full);
{Lấy 1 phần tử trong Buffer}
signal(empty);
{Xử lý phần tử vừa lấy ra}
} while (1);
```

Consumer

Consumer
Running
Producer
Running



Bài toán người sản xuất (producer)-người tiêu thụ(consumer)

Vấn đề: Khi có nhiều TT cùng loại Producer và Consumer => biến **IN** và **OUT**→ tài nguyên găng

Giải pháp: Dùng đèn báo thứ 3 (mutex) để diều độ TT cùng loại

```
do{
        {Tạo phần tử mới}
        wait(empty);
        wait(mutex);
        {Đặt phần tử mới vào Buffer}
        signal(mutex);
        signal(full);
} while (1);
```

Producer

```
do{
        wait(full);
        wait (mutex);
        {Lấy 1 phần tử trong Buffer}
        signal(mutex);
        signal(empty);
        {Xử lý phần tử vừa lấy ra}
} while (1);
```

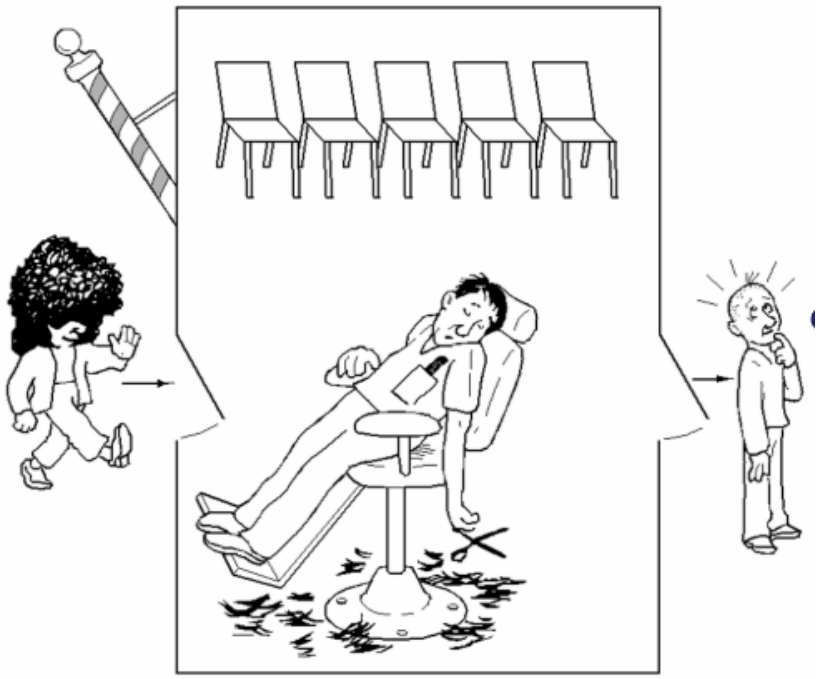
Consumer

Người đọc và biên tập viên

- Nhiều TT (Readers) cùng truy nhập 1 cơ sở dữ liệu (CSDL)
- Một số TT (Writers) cập nhật cơ sở dữ liệu
- Cho phép số lượng tùy ý các TT Readers cùng truy nhập CSDL
  - Đang tồn tại 1 TT Reader truy cập CSDL, mọi TT Readers khác mới xuất hiện đều được truy cập CSDL
  - (TT Writers phải xếp hàng chờ đợi)
- Chỉ cho phép 1 TT Writers cập nhật CSDL tại 1 thời điểm.
- Vấn đề không trưng dụng. Các TT ở trong đoạn găng mà không bị ngắt

Người thợ cắt tóc ngủ gật (Sleeping Barber)

- ***N*** ghế đợi dành cho khách hàng
- Một người thợ chỉ có thể cắt tóc cho một khách hàng tại một thời điểm
- Không có khách hàng đợi, thợ cắt tóc ngủ
- Khi một khách hàng tới
  - Nếu thợ cắt tóc đang ngủ⇒Đánh thức anh ta dậy làm việc
  - Nếu thợ cắt tóc đang làm việc
    - Không còn ghế đợi trống ⇒ bỏ đi
    - Còn ghế đợi trống⇒ Ngồi đợi



Bathroom Problem

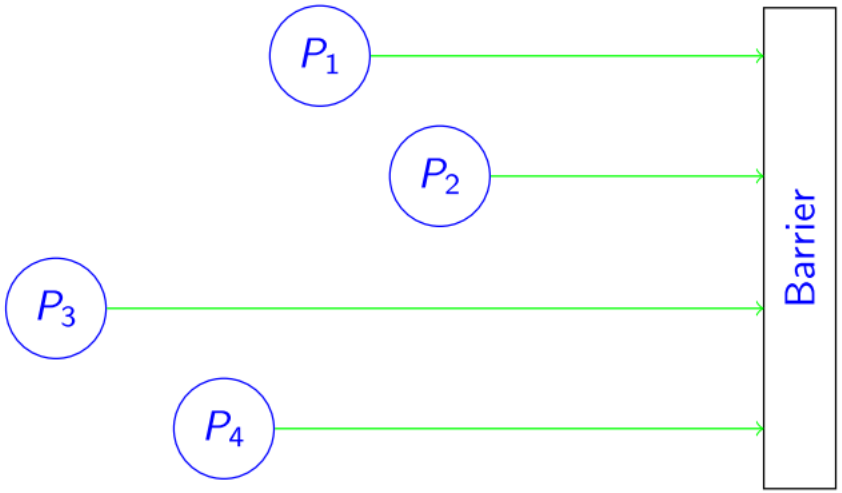
Bài toán

- A bathroom is to be used by both men and women, but not at the same time
- If the bathroom is empty, then anyone can enter
- If the bathroom is occupied, then only a person of the same gender as the occupant(s) may enter
- The number of people that may be in the bathroom at the same time is limited

- Yêu cầu cài đặt bài toán thỏa mãn các ràng buộc
  - Có 2 kiểu TT male() và female()
  - Mỗi TT ở trong Bathroom một khoảng t/gian ngẫu nhiên

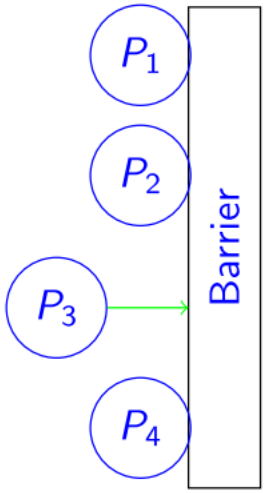
Đồng bộ barriers

- Các TT hướng tới một Ba-ri-e chung



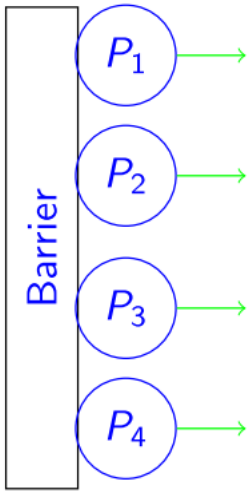
Đồng bộ barriers

- Các TT hướng tới một Ba-ri-e chung
- Khi đạt tới Ba-ri-e, tất cả các TT đều bị *block* ngoại trừ TT đến cuối cùng



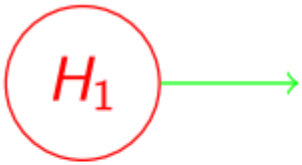
Đồng bộ barriers

- Các TT hướng tới một Ba-ri-e chung
- Khi đạt tới Ba-ri-e, tất cả các TT đều bị *block* ngoại trừ TT đến cuối cùng
- Khi TT cuối tới, đánh thức tất cả các TT đang bị block và cùng vượt qua Ba-ri-e



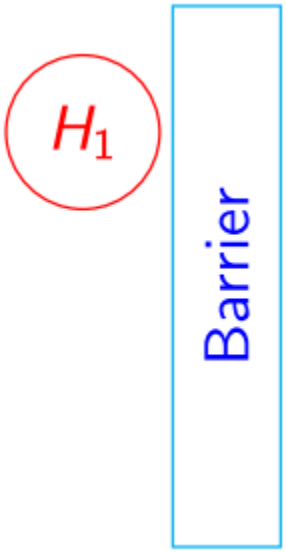
Bài toán tạo phân tử H2O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



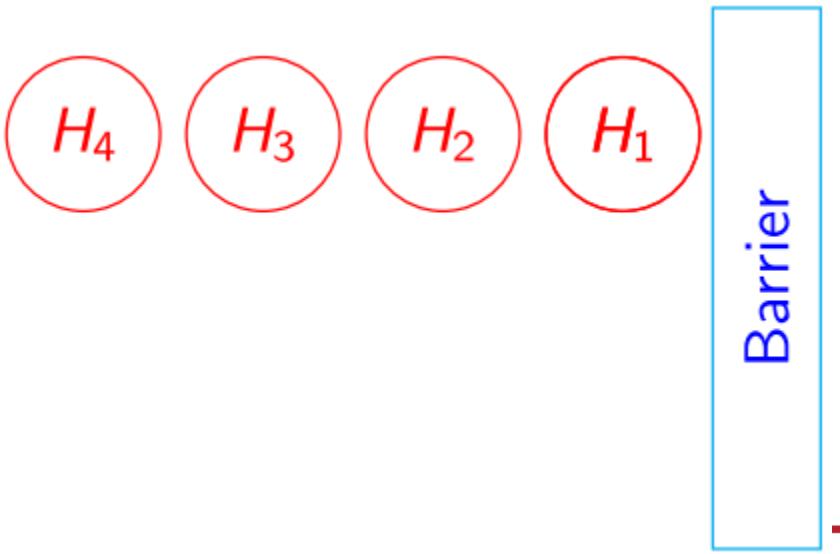
Bài toán tạo phân tử H2O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



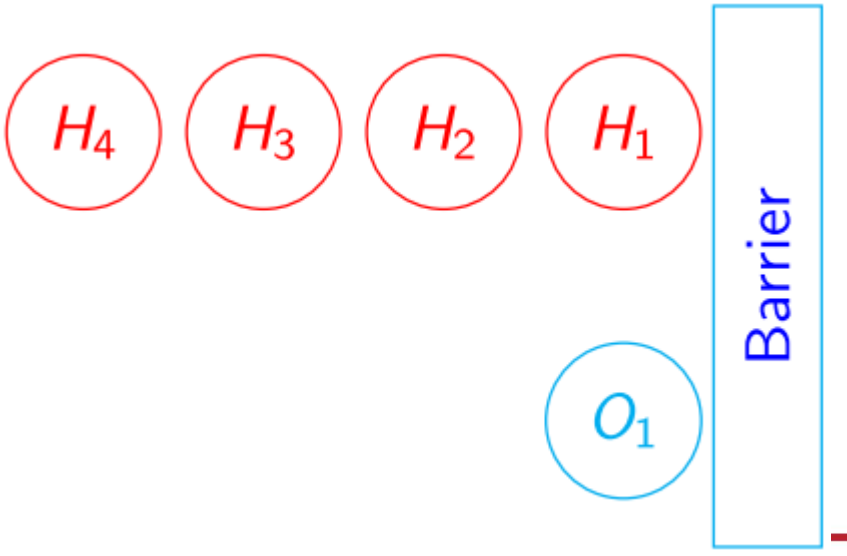
Bài toán tạo phân tử H2O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



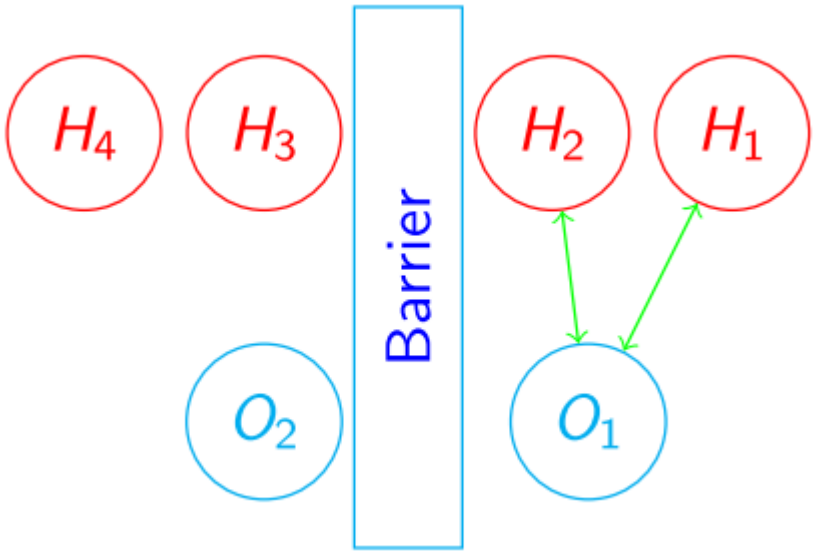
Bài toán tạo phân tử H2O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



Bài toán tạo phân tử H2O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



Bữa ăn tối của triết gia

Bài toán đồng bộ hóa tiến trình nổi tiếng, thể hiện tình trạng nhiều tiến trình phân chia nhiều tài nguyên

- 5 triết gia ăn tối quanh một bàn tròn
- Trước mỗi triết gia là một đĩa mì
- Giữa 2 đĩa kề nhau là một cái dĩa (fork)
- Các triết gia thực hiện luân phiên, liên tục 2 việc :Ăn và Nghĩ
- Mỗi triết gia cần 2 cái dĩa để ăn
- Chỉ lấy 1 dĩa tại 1 thời điểm
- Cái bên trái rồi tới cái bên phải
- Ăn xong, triết gia để dĩa vào vị trí cũ

Vấn đề triết gia ăn tối: Phương pháp đơn giản

- Mỗi chiếc dĩa là một tài nguyên găng, được điều độ bởi một đèn báo fork[i]
- Semaphore fork[5] = {1, 1, 1, 1, 1};
- Thuật toán cho Triết gia Pi

```
do{
 wait(fork[i])
     wait(fork[(i+1)% 5]);
          { Ăn}
     signal(fork[(i+1)% 5]);
 signal(fork[i]);
          {Nghĩ}
} while (1);
```

- Nếu tất cả các triết gia cùng muốn ăn
  - Cùng lấy chiếc dĩa bên trái (gọi tới: wait(fork[i]))
  - Cùng đợi lấy chiếc dĩa bên phải (gọi tới: wait(fork[(i+1)%5]))

⇒ Bế tắc (deadlock)

Vấn đề triết gia ăn tối: Giải pháp 1

- Chỉ cho phép một nhà triết học lấy dĩa tại một thời điểm
- Semaphore mutex ← 1;
- Thuật toán cho Triết gia Pi

```
do{
   wait(mutex)
       wait(fork[i])
       wait(fork[(i+1)% 5]);
   signal(mutex)
           { Ăn}
       signal(fork[(i+1)% 5]);
       signal(i);
           {Nghĩ}
} while (1);
```

- Có thể làm cho 2 triết gia không kề nhau cùng được ăn tại một thời điểm (P1: ăn, P2: chiếm mutex⇒ P3 đợi)

Vấn đề triết gia ăn tối: Giải pháp 2

- Thứ tự lấy dĩa của các triết gia khác nhau
- Triết gia số hiệu chẵn lấy dĩa trái trước
- Triết gia số hiệu lẻ lấy dĩa phải trước

```
do{
   j = i%2
   wait(fork[(i + j)%5])
       wait(fork[(i+1 - j)% 5]);
           { Ăn}
       signal(fork[(i+1 - j)% 5]);
   signal((i + j)%5);
           {Nghĩ}
} while (1);
```

- Giải quyết được vấn đề bế tắc

Vấn đề triết gia ăn tối: Một số giải pháp khác

- Trả lại dĩa bên trái nếu không lấy được cái bên phải
  - Kiểm tra dĩa phải sẵn sàng trước khi gọi wait(fork[(i+1)%5])
  - Nếu không sẵn có: trả lại dĩa trái, đợi một thời gian rồi thử lại
  - Không bị bế tắc, nhưng không tiến triển:nạn đói (starvation)
  - Thực hiện trong thực tế, nhưng không đảm bảo về lý thuyết của đề bài

Vấn đề triết gia ăn tối: Một số giải pháp khác (tiếp)

- Sử dụng đèn báo đồng thời P_{Sim}(S1, S2, . . . , Sn)
  - Thu được tất cả đèn báo cùng một thời điểm hoặc không có bất kỳ đèn báo nào
  - Thao tác P_{Sim}(S1, S2, . . . , Sn) sẽ block() TT/luồng gọi khi có bất kỳ một đèn báo nào không thể thu được
  - Thuật toán

```
P_Sim(fork[i], fork[(i+1)% 5]);

{ Ăn}
V_Sim(fork[i], fork[(i+1)% 5]);
```

  - Khó cài đặt đèn báo đồng thời
  - Giải pháp đề xuất bởi Tanenbaum (Tanenbaum 2001)

True problem ?