Đồ án Đồng Bộ Hóa

Môn: Hệ Điều Hành

Người thực hiện: Trần Văn Tú
Mã số Sinh Viên: 1810629

Mục lục

Contents

Mục lục

- Mức độ hoàn thiện : 100%
- Bài 1 : 100%
- Bài 2 : 100%
- Bài 3 : 100%

Bài 1

- Bài 1
- Chứng tỏ rằng giá trị của X có thể vượt quá 20.

Các TH X có thể vượt quá giá trị 20

Khi X = 19, hai lệnh X = X + 1 của 2 tiểu trình 1 và 2 chạy trước 2 lệnh so sánh

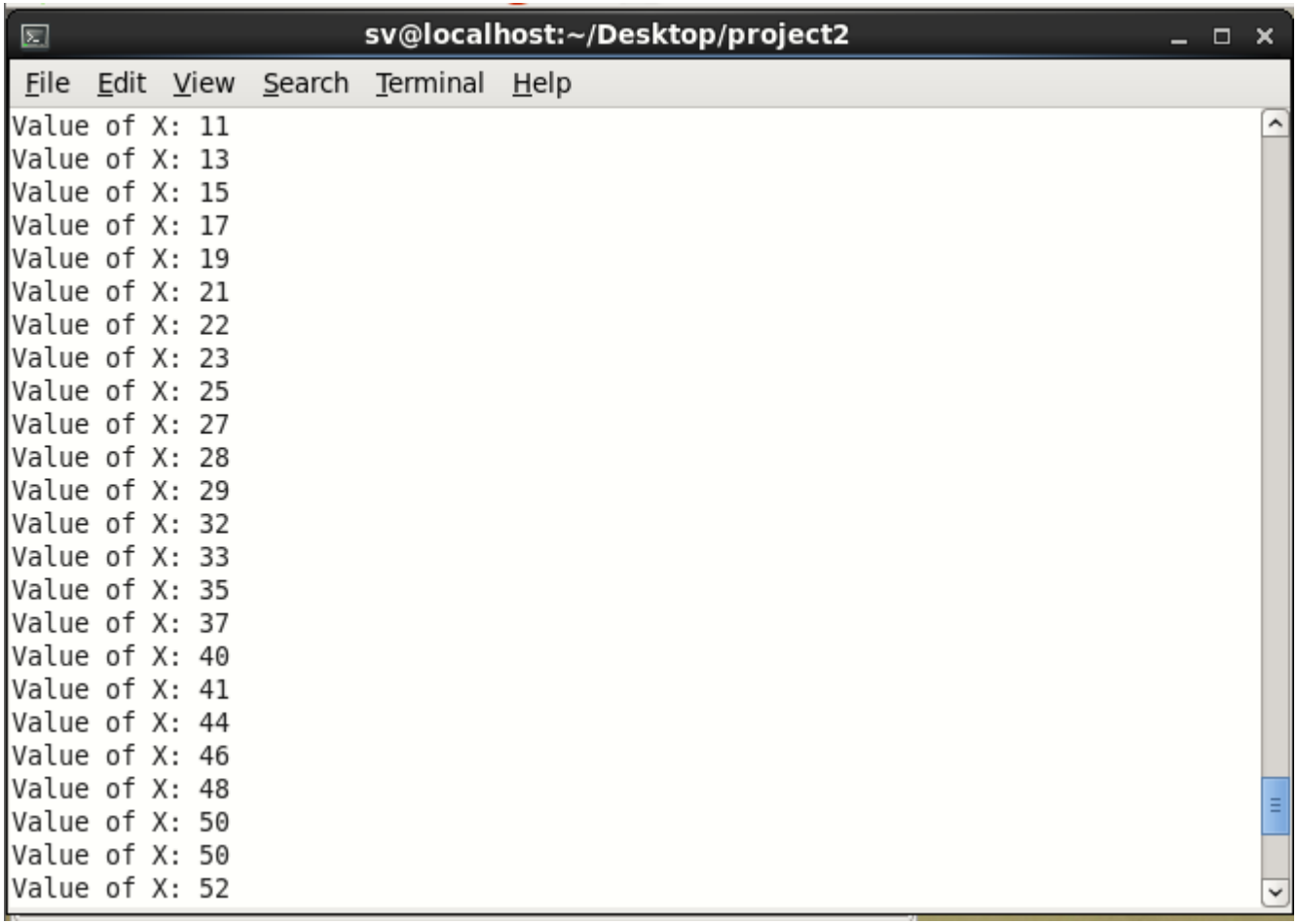
Có thể là lệnh X = X + 1 của tiểu trình 1 chạy trước lệnh X = X + 1 của tiểu trình 2, hoặc ngược lại, hoặc có thể chạy đồng thời, cùng thời điểm và yêu cầu là phải chạy trước 2 lệnh so sánh IF(X == 20) của 2 tiểu trình.

Tiểu trình 1	Tiểu trình 2	Giá trị của X theo thời gian
X = X + 1		20 = 19 + 1
	X = X + 1	21 = 20 + 1
IF (X == 20)		21
	IF(X == 20)	21

Bài 1

- Với giá trị đầu vào là X = 21 thì giá trị của X sẽ liên tục tăng, cho đến khi tràn số.

- Hình ảnh minh họa:



Bài 1

- Giải pháp đồng bộ: Thêm semaphore
- Bước 1: Xác định miền găng (critial section)

```
do{
        //Start critical section
        x=x+1;
        if(x==20)
            x=0;
        //End critical section
}while(1);
```

- Bước 2: Thêm semaphore
  - Khởi tạo semaphore với value ban đầu là 1 (đồng bộ độc quyền truy xuất)

```
sem_init(&mutex, 0, 1);
```

  - Thêm hàm semaphore wait (up) vào bắt đầu miền găng
  - Thêm hàm semaphore post (down) vào chỗ kết thúc miền găng

```
do{
        //printf("\n%d",x);
        sem_wait(&mutex);
        //critical section
        x=x+1;
        if(x==20)
           x=0;
        sem_post(&mutex);
        //usleep(100);
}while(1);
```

Bài 1

Bài 2

- Bài 2
- Kết quả sinh ra ngẫu nhiên:
  - Tạo 2 tiến trình P1, P2:
  - Chạy các tiểu trình:

```
A1
B1
A2
B2
[sv@localhost project2]$ ./bai2
A1
A2
B1
B2
[sv@localhost project2]$ ./bai2
B2
A2
A1
B1
[sv@localhost project2]$ ./bai2
B1
B2
A1
A2
[sv@localhost project2]$ ./bai2
A1
B1
B2
A2
[sv@localhost project2]$ ./bai2
B1
A2
B2
A1
[sv@locali project2 ect2]$
```

```
/* create 2 child processes */
 for (i = 0; i < 2; i++){
      pid = fork ();
      if (pid < 0)              /* check for error      */
          printf ("Fork error.\n");
      else if (pid == 0)
          break;                  /* child processes */
 }
```

Nhận xét: Các tiểu trình A1, A2, B1, B2 sẽ chạy với thứ tự ngẫu nhiên, do đó kết quả in ra sẽ ngẫu nhiên.

Bài 2

Giải pháp: Thêm semaphore

- Ý tưởng: Tiểu trình A2 sẽ phải chạy sau tiểu trình A1, B1.

  Tiểu trình B2 cũng sẽ phải chạy sau tiểu trình A1, B1.

- Cách làm: Để tiểu trình A2 chạy sau A1 thì chúng ta có thể sử dụng pthread_join() để chờ tiểu trình A1 kết thúc

  Để tiểu trình A2 chạy sau B1 thì phải sử dụng đến semaphore (đồng bộ theo kiểu hẹn hò), B1 kết thúc thì A1 mới được chạy.

Tương tự xử lý cho tiểu trình B2.

Tạo sem1 điều phối giữa A1 và B2, sem2 điều phối B1 và A2 (với value khởi tạo bằng 0).

```
int i;
sem_t *sem1, *sem2;
sem1 = sem_open ("pSem1", O_CREAT | O_EXCL, 0644, value);
sem2 = sem_open ("pSem2", O_CREAT | O_EXCL, 0644, value);
/* name of semaphore is "pSem", semaphore is reached using this name */
sem_unlink ("pSem1");
sem_unlink ("pSem2");
```

Đoạn code trong tiểu trình:

```
if(i%2==0){
        pthread_create(&t1,NULL,A1,NULL);
        pthread_join(t1, NULL);
        sem_post(sem1);
        sem_wait(sem2);
        pthread_create(&t2,NULL,A2,NULL);
        pthread_join(t2, NULL);
} else{
        pthread_create(&t1,NULL,B1,NULL);
        pthread_join(t1, NULL);
        sem_post(sem2);
        sem_wait(sem1);
        pthread_create(&t2,NULL,B2,NULL);
        pthread_join(t2, NULL);

}
```

Bài 2

- Kết quả:
  - Các tiến trình A1, B1 sẽ chạy trước tiến trình A2, B2

```
A1
B1
A2
B2
[sv@localhost project2]$ ./bai2_sem
B1
A1
A2
B2
[sv@localhost project2]$ ./bai2_sem
A1
B1
B2
A2
[sv@localhost project2]$ ./bai2_sem
A1
B1
B2
A2
[sv@localhost project2]$ ./bai2_sem
A1
B1
B2
A2
[sv@localhost project2]$ ./bai2_sem
A1
B1
A2
B2
```

Bài 3

- Bài 3
- In ra vô hạn H20, H, O.

- Bước 1: Tạo vòng lặp while(1) để in ra vô hạn tiểu trình

```
pthread_t t1,t2,t3;
while(1){
        pthread_create(&t1,NULL,MakeH,NULL);
        pthread_create(&t2,NULL,MakeO,NULL);
        pthread_create(&t3,NULL,MakeH2O,NULL);
}
return 0;
```

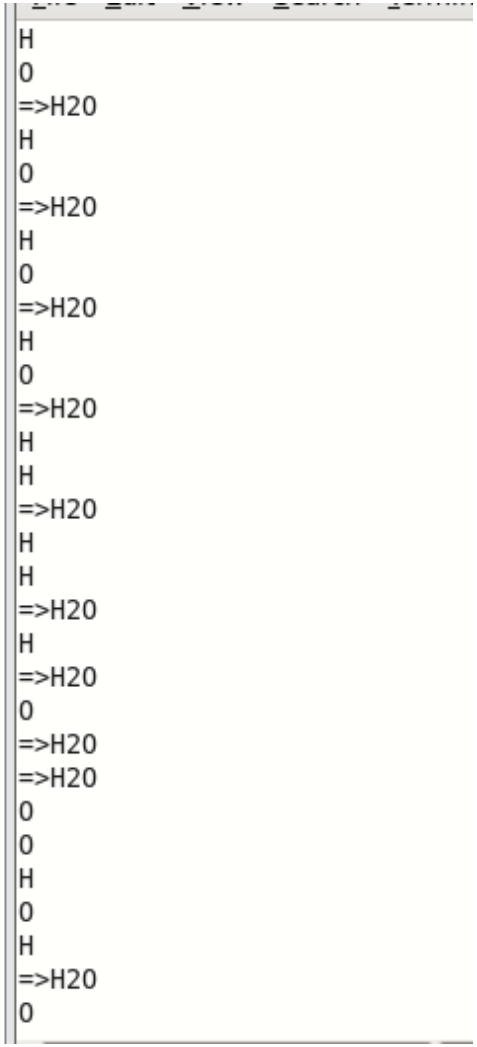
- Bước 2: thêm detach để hủy tiểu trình

```
void* MakeH(void* arg) {
        printf("H\n");
        pthread_detach(pthread_self());
}

void* MakeO(void* arg) {
        printf("O\n");
        pthread_detach(pthread_self());
}
void* MakeH2O(void* arg) {
        printf("=>H2O\n");
        pthread_detach(pthread_self());
}
```

Bài 3

- Kết quả:


H
0
=>H20
H
0
=>H20
H
0
=>H20
H
0
=>H20
H
H
=>H20
H
H
=>H20
H
=>H20
0
=>H20
=>H20
0
0
H
0
H
=>H20
0

Bài 3

❑ Giải pháp đồng bộ: thêm semaphore

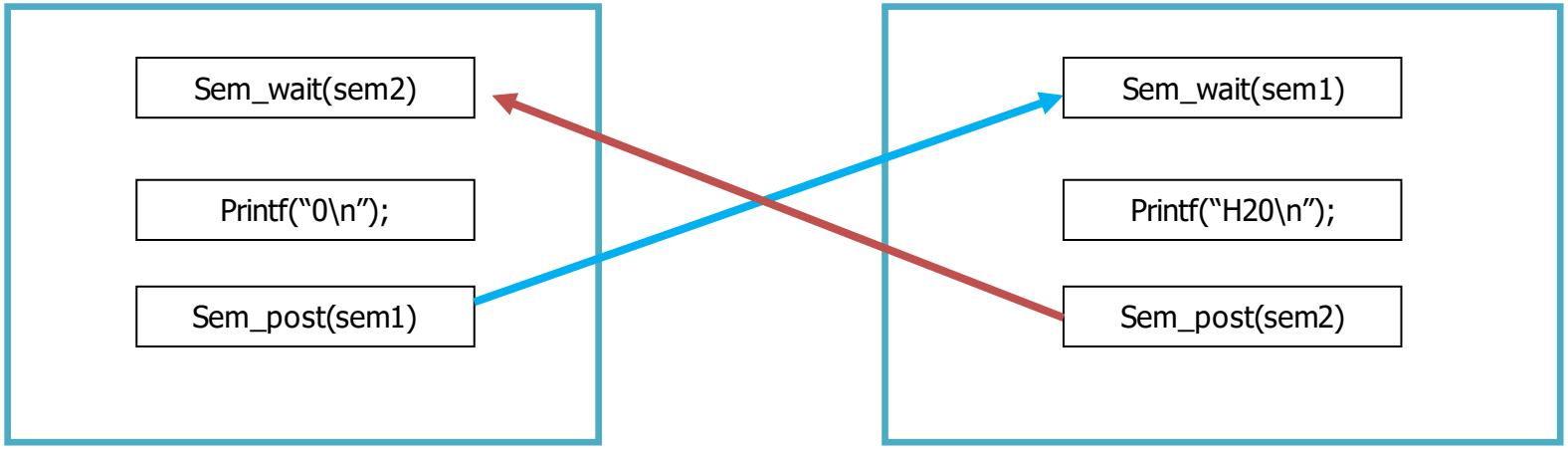
- Ý tưởng: + Giữa O và H2O, điều phối sẽ chạy theo thứ tự: O →H20 → O → H20 → O → H2O …→ O → H20 …

  + Giữa H vào H2O, sau khi điều phối sẽ chạy theo thứ tự: H → H →H20 →H → H → H20 … → H20 → H → H …

  + Kết hợp 2 kết quả trên sẽ được kết quả cần tìm: O →H → H → H20 → H→ O → H → H20 …

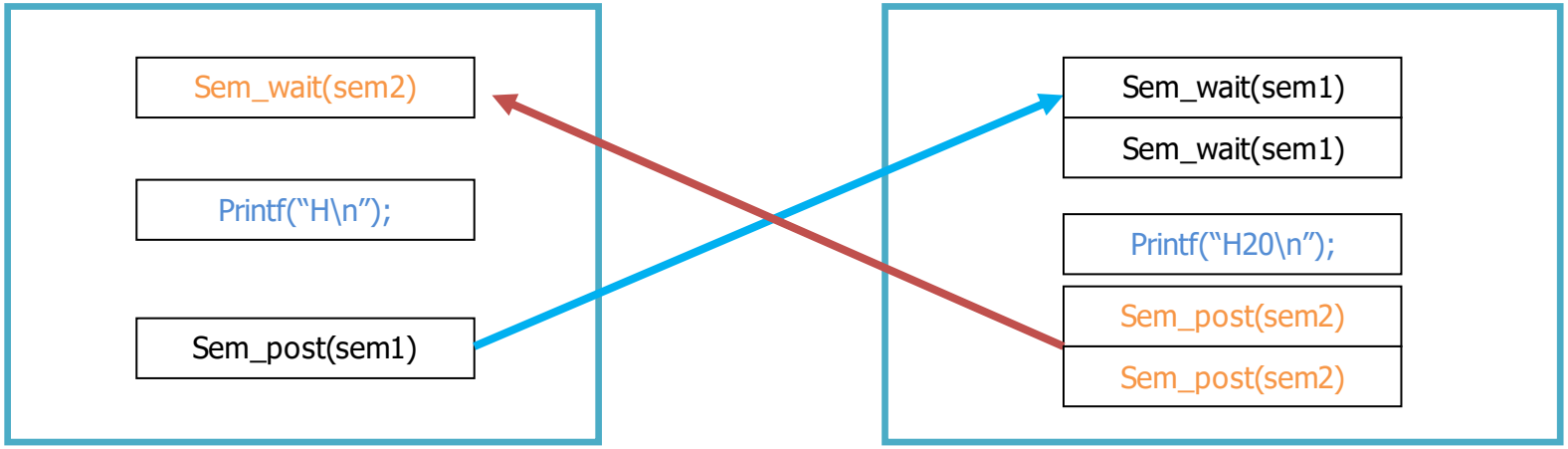
- Cách làm:

  + Để điều phối giữa O và H20 thì cần 2 semaphore như sau:
  Sem 1 với value là 0, và sem 2 với value là 1



Bài 3

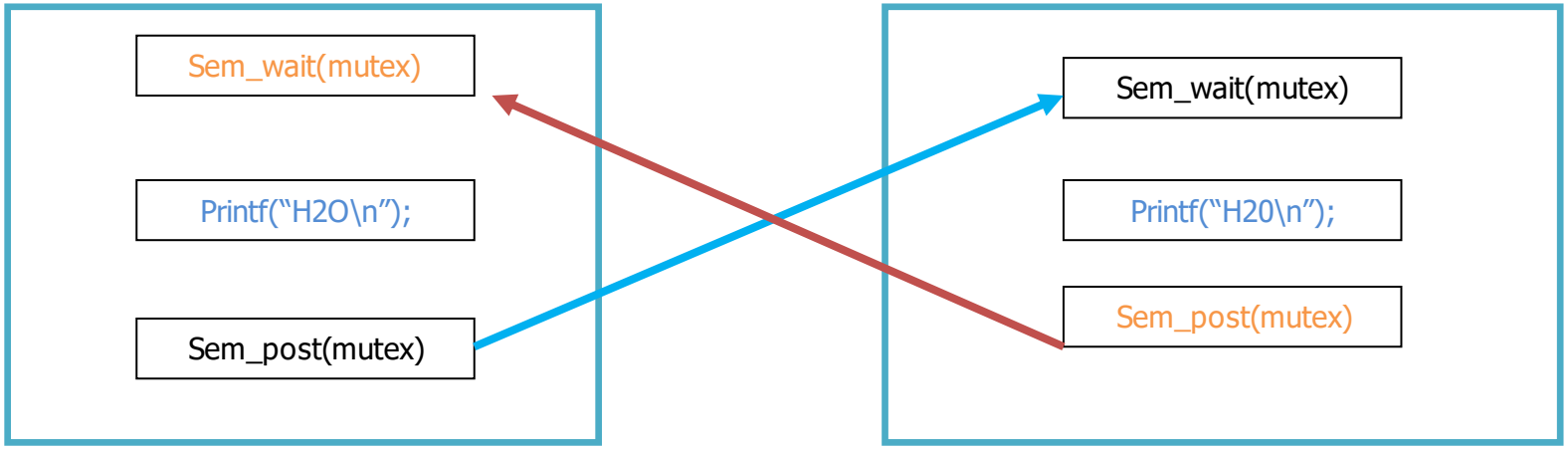
+ Để điều phối giữa H và H20 thì cần 2 semaphore như sau:
Sem 1 với value là 0, và sem 2 với value là 2 và



+ Tuy nhiên lúc này sẽ xuất hiện vấn đề mất đồng bộ giữa các tiểu trình H20 do đó phải thêm 1 semaphore nữa để truy xuất độc quyền.

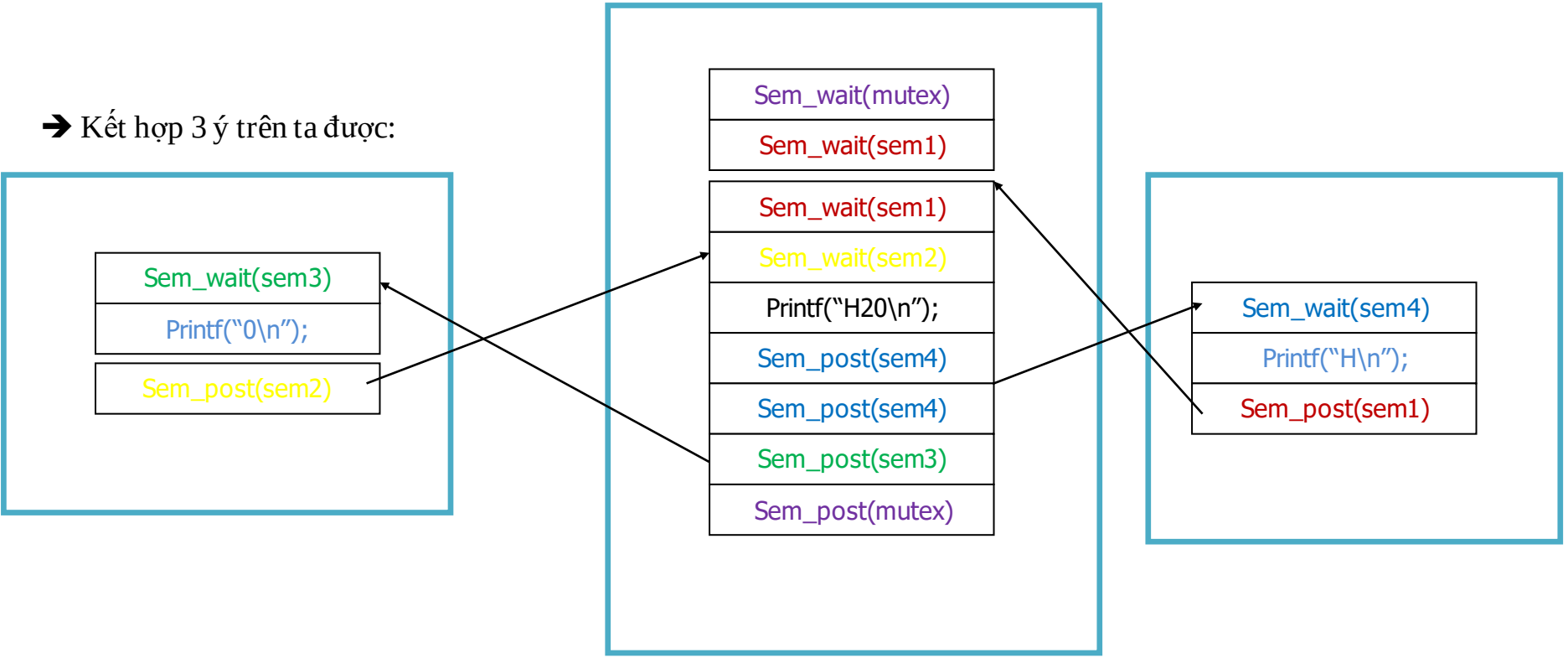
Khởi tạo thêm semaphore mutex với value = 1 (Tại 1 thời điểm chỉ có thể có 1 H20 vào miền găng).

Bài 3


Sem_wait(mutex)
Printf("H2O\n");
Sem_post(mutex)
Sem_wait(mutex)
Printf("H20\n");
Sem_post(mutex)

Bài 3

➔ Kết hợp 3 ý trên ta được:



Semaphore	Value
Sem1	0

Bài 3

Sem2	0
Sem3	1
Sem4	2
mutex	1

- Kết quả:

```
...
H
O
H
=>H2O
...
H
H
O
=>H2O
...
H
H
O
=>H2O
...
H
H
O
=>H2O
...
O
H
H
```