60 phút

[ARC] NEW LTPT

* Bắt buộc

* Biểu mẫu này sẽ ghi lại tên của bạn, vui lòng điền vào tên của bạn.

1

Các tiến trình (Process) trong một hệ thống phân tán, gồm nhiều máy tính đặt tại nhiều địa điểm khác nhau, sử dụng mô hình nào để giao tiếp với nhau? * (2 Điểm)

- [ ] Mô hình truyền thông điệp qua mạng truyền thông
- [ ] KHÔNG sử dụng Mô hình bộ nhớ chia sẻ, cũng như Mô hình truyền thông điệp qua mạng truyền thông
- [ ] Mô hình bộ nhớ chia sẻ

2

Đâu là những đặc điểm chung của các thuật toán sử dụng tài nguyên phụ token (như **Thuật toán Tập trung, Thuật toán Vòng tròn Token**) cho bài toán truy cập tài nguyên chia sẻ (i.e., khu vực quan trọng) trong một hệ thống phân tán ? * (2 Điểm)

- [ ] Tất cả các thuật toán này đều sử dụng 1 kiểu thông điệp cho một lần đi vào khu vực quan trọng: thông điệp token
- [ ] Các thuật toán này LUÔN đảm bảo được yêu cầu về tính công bằng, tức là tiến trình nào yêu cầu đi vào khu vực quan trọng trước thì sẽ được ưu tiên đi vào khu vực quan trọng trước
- [ ] Tiến trình nào giữ token sẽ được quyền đi vào khu vực quan trọng, và tại một thời điểm chỉ có một tiến trình giữ token

3

Trạng thái đua tranh(race condition) giữa các luồng dẫn đến những điều nào sau đây ? * (2 Điểm)

- [ ] Dữ liệu chia sẻ có thể mất mát khi các luồng cùng thực hiện việc thay đổi dữ liệu đó
- [ ] Tính chính xác của chương trình bị phụ thuộc vào thời gian thực thi tương đối của các sự kiện
- [ ] Giá trị của các biến chia sẻ luôn luôn nhất quán và chính xác khi các luồng thực hiện việc cập nhật các biến đó

4

Mục đíchcủa dòng lệnh if(numberReaders ==1)wlock.P(); trong phương thức startRead() là gì * (2 Điểm)

- [ ] Đánh thức một luồng ghi bất kỳ đang bị khóa dậy để thực hiện tiếp công việc của nó
- [ ] Nếu đây là luồng đọc đầu tiên thành công đi vào khu vực quan trọng thì sẽ chuyển sang semaphore nhị phân sang wlock sang trạng thái không sẵn sáng để chặn các luồng ghi phía sau đi vào khu vực quan trọng
- [ ] Đánh thức một luồng đọc bất ký đang bị khóa dậy để thực hiện tiếp công việc của nó

5

Trong bài toán người đọc, người ghi với n>1 luồng đọc & m>1 luồng ghi cùng hoạt động, chúng ta cần đảm bảo những điều kiện đồng bộ nào * (2 Điểm)

- [ ] Ràng buộc đọc-ghi: Một luồng độc và một luồng ghi không được truy cập đồng thời vào CSDL chia sẻ
- [ ] Ràng buộc đọc-đọc. Hai luồng đọc không được truy cập đồng thời vào CSDL chia sẻ
- [ ] Ràng buộc ghi-ghi: hai luồng không được truy cập đồng thời vào CSDL chia sẻ
- [ ] Nhiều luồng đọc có thể đồng thời truy cập CSL chia sẻ

6

Semaphore đếm khác Semaphore nhị phân như thế nào ? * (2 Điểm)

- [ ] Không có hai thao tác P(), V() trong Semaphore đếm
- [ ] Không có hàng đợi các luồng bị khóa trong Semaphore đếm
- [ ] Semaphore đếm có thể được dùng để cho phép nhiều luồng đồng thời cùng ở trong khu vực quan trọng (CS), trong khi Semaphore nhị phân chỉ cho phép nhiều nhất 1 luồng ở trong CS tại một thời điểm
- [ ] Biến value của Semaphore đếm có thể nhận nhiều giá trị lớn hơn 0, trong khi biến value của Semaphore nhị phân chỉ có thể nhận 2 giá trị

7

Xét thuật toán trong Hình dưới để giải quyết bài toán loại trừ lẫn nhau trong một chương trình đồng thời có 2 luồng cùng thực thi.
Những kịch bản thực thi nào của 4 câu lệnh (được đánh số màu đỏ), trong 2 phương thức **requestCS()**, sẽ khiến cho chương trình rơi vào **trạng thái khoá chết** (deadlock), tức là cả hai luồng đều bị tắc lại ở vòng lặp while, không luồng nào đi vào được khu vực quan trọng ? * (2 Điểm)

T_0

```
class Attempt2 implements Lock {
   boolean wantCS[] = {false, false};
   public void requestCS(int i) {
1     wantCS[i] = true;
2     while (wantCS[1 - i]) ;
   }
   public void releaseCS(int i) {
      wantCS[i] = false;
   }
}
```

T_1

```
class Attempt2 implements Lock {
   boolean wantCS[] = {false, false};
   public void requestCS(int i) {
      wantCS[i] = true;          3
      while (wantCS[1 - i]) ;    4
   }
   public void releaseCS(int i) {
      wantCS[i] = false;
   }
}
```

- [ ] 3->4->2->1
- [ ] 1->3->4->2
- [ ] 3->1->4->2
- [ ] 3->1->2->4
- [ ] 1->2->3->4
- [ ] 1->3->2->4

8

Khi chạy chương trình sau có thể sinh ra những kết quả nào ? * (2 Điểm)

```
public class Lab1 extends Thread{
    private int id;
    public Lab1(int _id) {
        id = _id;
    }
    public void run() {
            System.out.print("T-" + id + " ");
    }

    public static void main(String[] args) throws InterruptedException {
        Lab1 t1 = new Lab1(1);
        Lab1 t2 = new Lab1(2);

        t1.start();
        t2.start();

        t2.join();

        System.out.print("T-m ");
    }
}
```

- [ ] T-1 T-m T-2
- [ ] T-1 T-2 T-m
- [ ] T-m T-2 T-1
- [ ] T-m T-1 T-2
- [ ] T-2 T-1 T-m
- [ ] T-2 T-m T-1

9

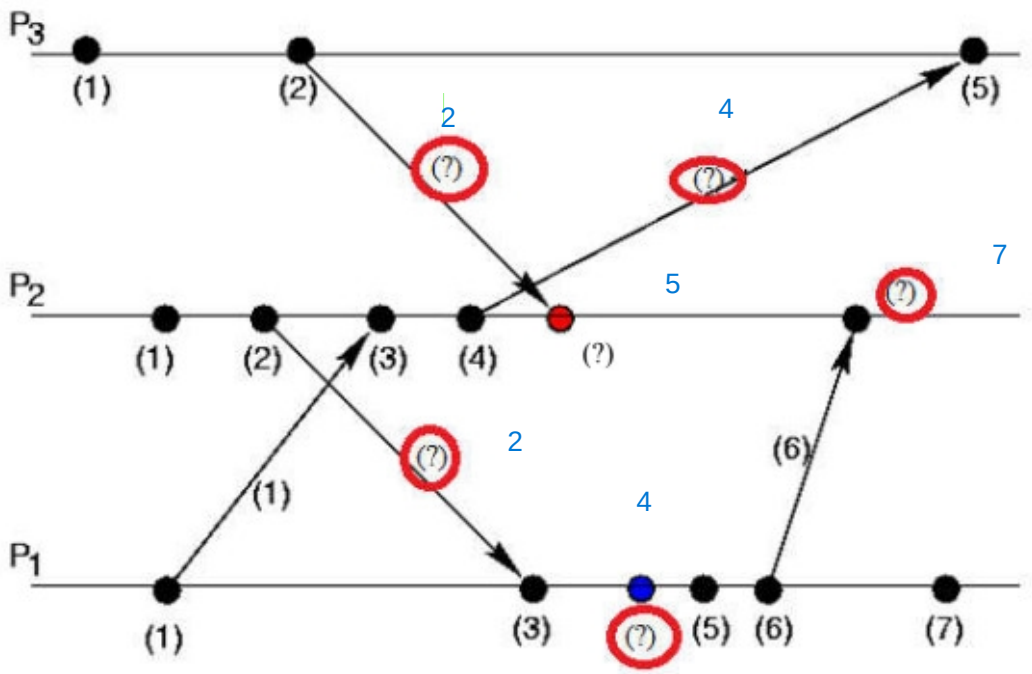
Giả sử các câu lệnh được thực thi một các nguyên tử. Sau khi 2 luồng t,u thực thi xong các câu lệnh của mình, biến chia sẻ counter có thể nhận những giá trị nào sau đây * (2 Điểm)

int counter = 0;	
Luồng *t*	Luồng *u*
int cnt;	int cnt;
cnt = counter; counter = cnt + 1;	cnt = counter; counter = cnt + 1;

- [ ] 0
- [ ] 1
- [ ] 3
- [ ] 2

10

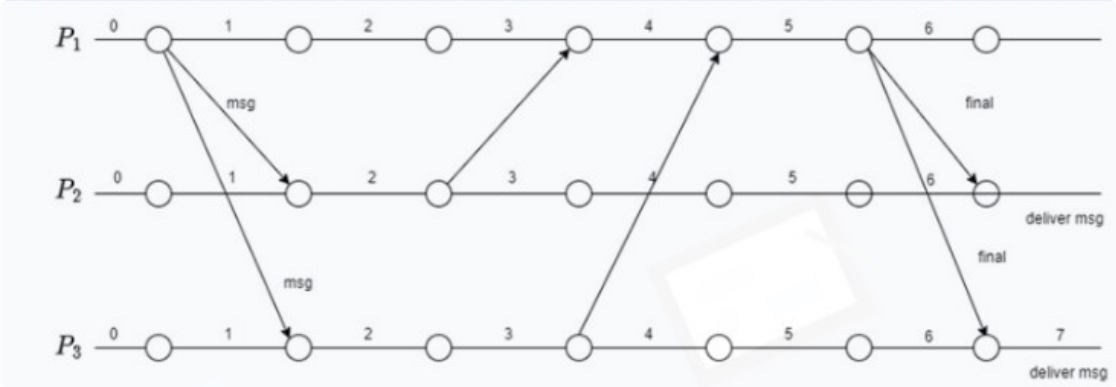
Đồng hồ Logic * (2 Điểm)



◯ Click đây

11

Cho sơ đồ tiến trình - thời gian như hình dưới đây, trong đó chúng ta sử dụng đồng hồ logic để đánh dấu thời gian của các trạng thái.
Giả sử tại một thời điểm nào đó, tiến trình P0 gửi một thông điệp đa hướng **msg** tới 2 tiến trình P1 và P2.
Sử dụng thuật toán Skeen để sắp thứ tự toàn bộ cho các thông điệp đa hướng trong hệ thống phân tán trên.
Dấu thời gian của thông điệp **msg** sau khi hoàn tất các bước của thuật toán Skeen là bao nhiêu? * (2 Điểm)



- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5

12

Các luồng (Thread) trong hệ thống đồng thời dựa trên cơ chế hóa, chạy trên một máy tính gồm nhiều bộ vi sử lý giao tiếp với nhau bằng cách nào? * (2 Điểm)

- [ ] Sử dụng bộ nhớ chia sẻ
- [ ] Sử dụng mô hình truyền thông điệp thông qua mạng truyền thông
- [ ] Không sử dụng mô hình bộ nhớ chia sẻ, cũng như mô hình truyền thông điệp qua mạng truyền thông

13

Ưu điểm của việc sử dụng các cấu trúc đồng bộ hoá (như Semaphore, Monitor) cho bài toán loại trừ lẫn nhau trong các chương trình đồng thời, so với các thuật toán sử dụng vòng lặp để kiểm tra điều kiện đi vào khu vực quan trọng như Peterson, Dekker, Bakery, là gì ? * (2 Điểm)

- [ ] Giải quyết được vấn đề bận chờ (busy-waiting), không gây lãng phí chu trình CPU
- [ ] Không có ưu điểm gì hơn so với các thuật toán đó
- [ ] Thời gian chạy chương trình nhanh hơn

14

Những phát biểu nào sau đây là đúng về khái niệm Socket dùng để phát triển các ứng dụng phân tán ? * (2 Điểm)

- [ ] Lập trình Socket CHỈ có thể được thực hiện dựa trên giao thức UDP (Universal Datagram Protocol)
- [ ] Socket là đối tượng được sử dụng để gửi và nhận thông điệp giữa các tiến trình trong một hệ thống phân tán
- [ ] Socket gồm 2 thành phần: Địa chỉ IP, Cổng
- [ ] Socket cung cấp giao diện ở mức thấp cho việc xây dựng các chương trình phân tán

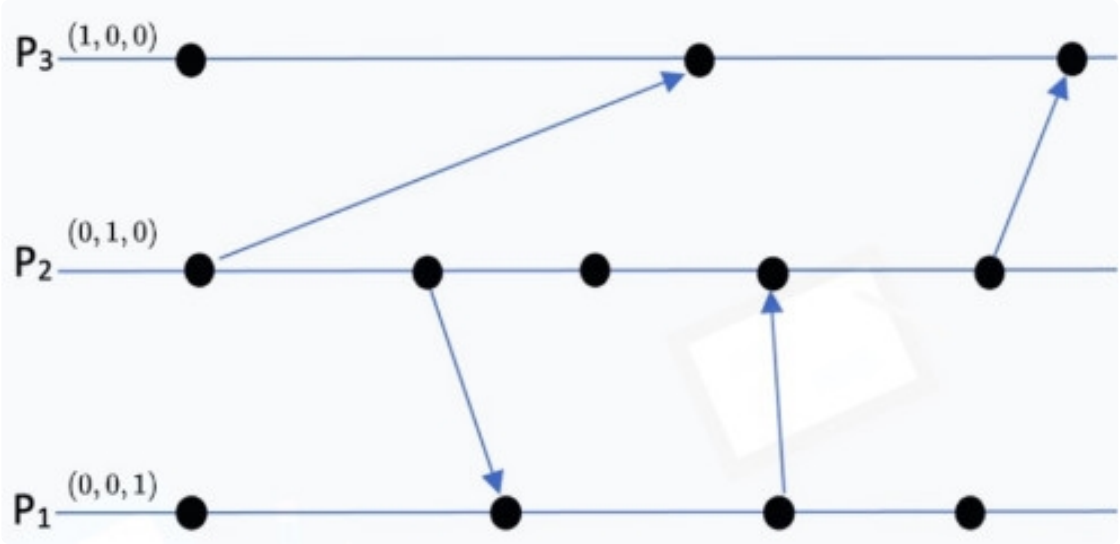
15

Những phát biểu đúng về RMI * (2 Điểm)

- [ ] Mỗi lời gọi từ xa luôn được thực hiện thông qua 2 đối tượng đại diện: stub ở phía client & skelenton ở phía sever
- [ ] Trong RMI, tiến trình gửi yêu cầu không cần biết đến vị trí thực sự của đối tượng từ xa
- [ ] Hai đối tượng stub & skeleton không bắt buộc phải được tạo ra khi thực hiện một lời gọi từ xa
- [ ] Trong RMI, chúng ta có thêm một thành phần trung gian RMIRegistry, giúp đăng ký và lấy về tham chiếu các đối tượng từ xa

16

Cho sơ đồ tiến trình - thời gian, hoặc sơ đồ đã-xảy-ra-trước, như hình vẽ dưới đây.
Sử dụng **cơ chế đồng hồ phụ-thuộc-trực-tiếp** để đánh dấu thời gian của các trạng thái cho các tiến trình.
Dấu thời gian của trạng thái cuối cùng, sau khi sự kiện cuối cùng xảy ra, của **tiến trình P3** là gì? * (2 Điểm)



- [ ] (3,6,0)
- [ ] (3,5,0)
- [ ] (4,2,0)
- [ ] (0,5,6)

17

Số lượng khu vực quan trọng (CS) của một luồng, trong các chương trình đồng thời, có thể là bao nhiêu * (2 Điểm)

- [ ] 0
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] Không giới hạn

18

Cho bài toán Người đọc - Người ghi, trong đó có n > 2 luồng đọc và m > 2 luồng ghi, cùng tương tác với cơ sở dữ liệu chia sẻ
Nếu các khối lệnh cho việc <<Ghi dữ liệu vào DB>> và <<Đọc dữ liệu từ DB>> được chuyển vào bên trong Monitor (trong khối synchronized) thì điều gì xảy ra? * (2 Điểm)

int numReader, numWriter; Object object;

Writer	Reader

void writeDB() {
synchronized (object) {
while (numReader > 0 ||
numWriter > 0)
object.wait();
numWriter = 1;
}
<< GHI DỮ LIỆU VÀO DB >>
(KHÔNG Ở TRONG MONITOR);
 	
synchronized (object) {
numWriter = 0;
object.notifyAll();
}
}
	void readDB() {
	synchronized (object) {
	while (numWriter > 0)
	object.wait();
	numReader++;
	}
	<< ĐỌC DỮ LIỆU TỪ DB ??
	(KHÔNG Ở TRONG MONITOR);
	
	synchronized (object) {
	numReader--;
	object.notify();
	}
	}

- [ ] Việc này sẽ chỉ cho phép nhiều nhất một luồng được thực hiện việc đọc dữ liệu từ csdl chia sẻ tại một thời điểm bất kỳ -> không thỏa mãn được yêu cầu cho phép nhiều luồng đọc cùng đọc csdl chia sẻ
- [ ] Thuật toán vẫn hoạt động thỏa mãn 3 yêu cầu đồng bộ hóa của bài toán Người đọc - Người ghi
- [ ] Việc này sẽ khiến cho việc sử dụng biến chia sẻ numWriter, numReader không còn cần thiết nữa

19

Xét một bài toán cho bài toán Sản xuất & Tiêu thụ, sử dụng cấu trúc đồng bộ hóa Semaphore như hình
Dòng lện isEmpty().V(), trong những phương thức deposit() được dùng mục đích gì * (2 Điểm)

BinarySemaphore **mutex**(true);
CountingSemaphore **isFull**(size), **isEmpty**(0)

Producer	Consumer

void **deposit**() {
isFull.P();
mutex.P();
<Ghi dữ liệu vào bộ đệm>
mutex.V();
isEmpty.V();
}
	void **fetch**() {
	isEmpty.P();
	mutex.P();
	<Lấy dữ liệu ra khỏi bộ đệm>
	mutex.V();
	isFull.V();
	}

- [ ] Đánh thức luồng Producer dậy để thực thi tiếp công việc của nó
- [ ] Khóa luồng Consumer
- [ ] Khoá luồng Producer
- [ ] Đánh thức luồng Consumer để thực thi công việc của nó

20

Semaphore nhị phân gồm những thành phần nào * (2 Điểm)

- [x] Thao tác P() được thực thi nguyên tử: dùng để thêm luồng gọi vào hàng đợi nếu semaphore không ở trạng thái sẵn sàng
- [x] Một biến value kiểu boolean
- [x] Một hàng đợi các luồng bị khóa - được khởi tạo là rỗng
- [x] Thao tác V() được thực thi nguyên tử: dùng để đánh thức một luồng bất kỳ trong hàng đợi
- [ ] Một biến value kiểu int

21

Khu vực quan trọng CS (Critical Region hay Critical Section) của một luồng, trong các chương trình đa luồng, là gì ? * (2 Điểm)

- [ ] Phần mã nguồn thực hiện một tính toán quan trọng của luồng
- [x] Phần mã nguồn của luồng cần được thực thi một cách nguyên tử
- [ ] Phần mã nguồn của luồng chia sẻ với một luồng khác
- [ ] Phần dữ liệu chia sẻ giữa các luồng

22

Đâu là những đặc điểm chung của các thuật toán sử dụng dấu thời gian (như **Thuật toán của Lamport, Thuật toán của Ricart & Agrawala**) cho bài toán truy cập tài nguyên chia sẻ (i.e., khu vực quan trọng) trong một hệ thống phân tán ? * (2 Điểm)

- [x] Các thuật toán này đảm bảo được yêu cầu về tính công bằng do sử dụng dấu thời gian của các thông điệp để quyết định thứ tự đi vào khu vực quan trọng
- [ ] Tất cả các thuật toán này đều sử dụng 3 kiểu thông điệp cho một lần đi vào khu vực quan trọng: thông điệp request, thông điệp ack, thông điệp release
- [x] Khi muốn đi vào khu vực quan trọng, một tiến trình đầu tiên phải gửi thông điệp yêu cầu (thông điệp request), có gắn dấu thời gian, tới tất cả tiến trình khác

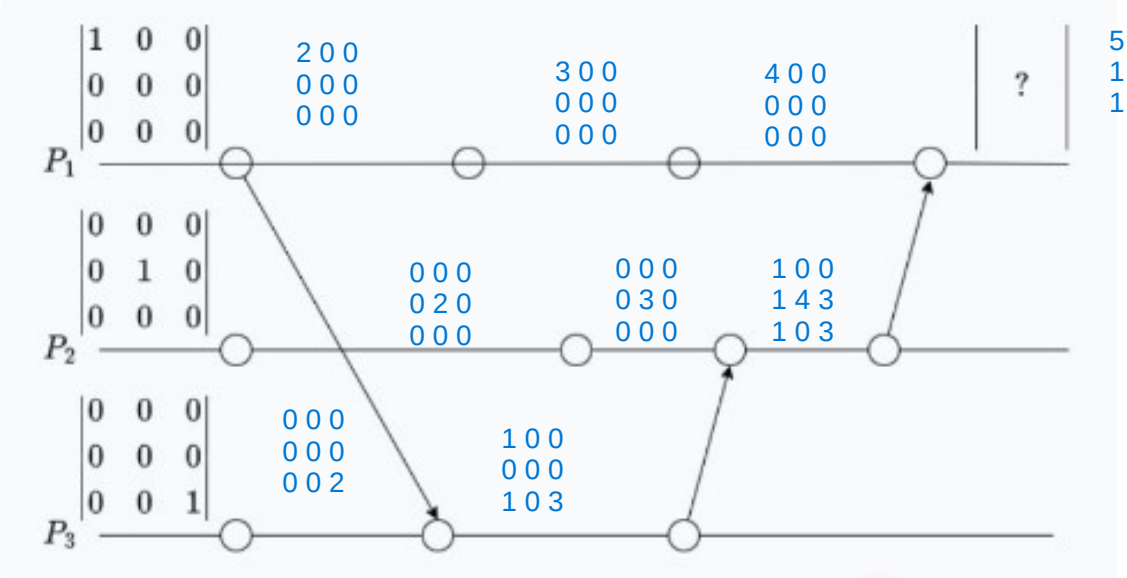
23

Những phát biểu nào sau đây đúng khi khái quát về hệ thống phân tán * (2 Điểm)

- [ ] Các tiến trình trong một hệ thống phân tán luôn biết được trạng thái toàn cục của hệ thống tại bất kỳ thời điểm nào
- [x] Không tồn tại biến chia sẻ giữa các tiến trình trong một hệ thống phân tán
- [x] Các tiến trình trong một hệ thống phân tán giao tiếp với nhau bằng cách gửi & nhận thông điệp qua mạng truyền thông
- [ ] Mỗi tiến trình trong hệ thống phân tán chỉ gồm có một luồng

24

Câu hỏiCho sơ đồ tiến trình - thời gian, hoặc sơ đồ đã-xảy-ra-trước, như hình vẽ dưới đây.
Sử dụng **cơ chế đồng hồ ma trận** để đánh dấu thời gian của các trạng thái cho các tiến trình.
Dấu thời gian của trạng thái cuối cùng, sau khi sự kiện cuối cùng xảy ra, của **tiến trình P1** là gì? * (2 Điểm)



- [ ] |4 4 3, 1 4 3, 1 0 3|
- [ ] |5 4 3, 1 4 3, 1 0 2|
- [ ] |5 4 3, 1 4 3, 1 0 3|
- [ ] |5 4 3, 1 3 3, 1 0 2|

25

Thuật toán Bakery của Lamport cho bài toán loại trừ lẫn nhau trong các chương trình đồng thời có thể hoạt động với bao nhiêu luồng * (2 Điểm)

- [ ] Tối đa 4 luồng
- [ ] Chính xác 2 luồng
- [ ] Tối đa 3 luồng
- [ ] Số lượng luồng bất kỳ lớn hơn 1

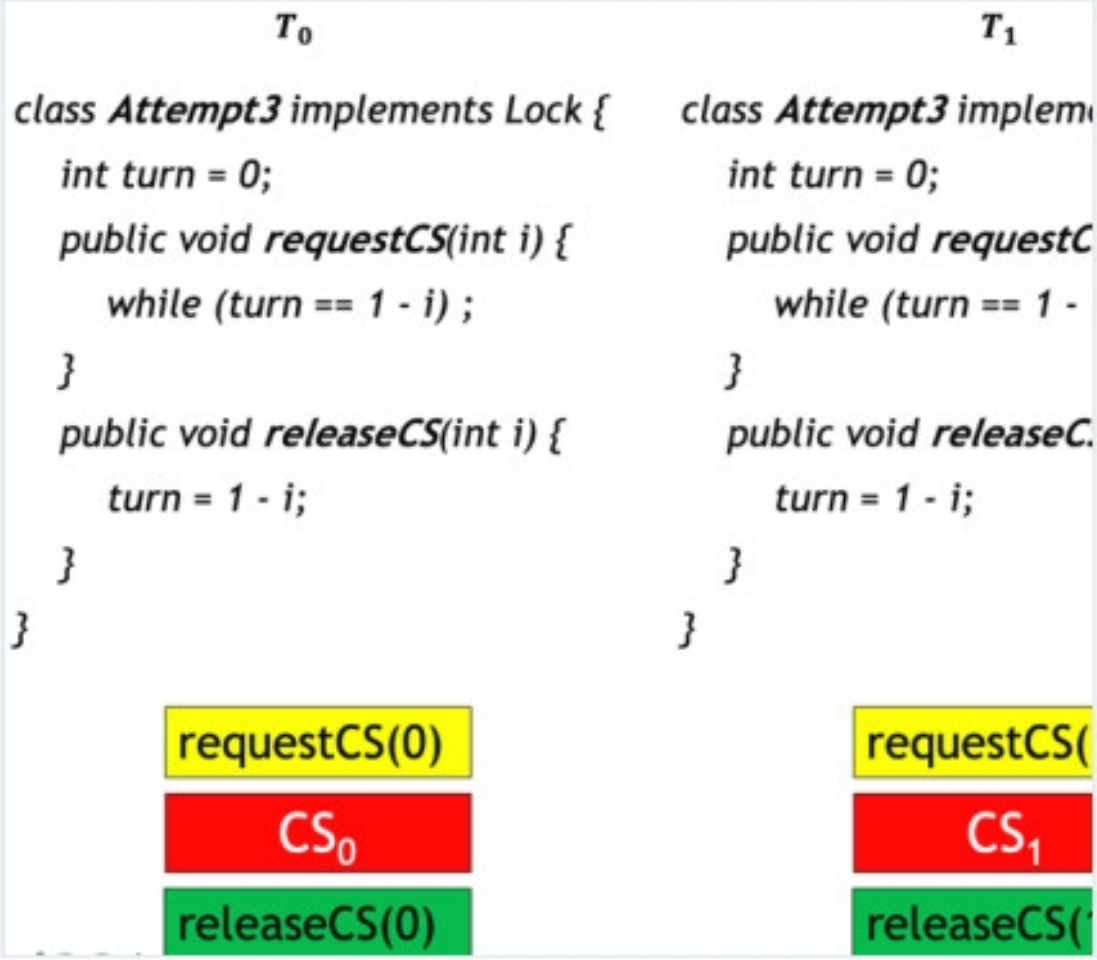
26

Nhược điểm chung của các thuật toán Peterson, Dekker, Bakery khi giải quyết bài toán loại trừ lẫn nhau trong các chương trình đồng thời là gì? * (2 Điểm)

- [ ] Các luồng phải liên tục kiểm tra xem điều kiện đi vào khu vực quan trọng đã được thoả mãn hay chưa, thông qua vòng lặp. Điều này dẫn đến gây lãng phí chu trình CPU
- [ ] Sử dụng các biến chia sẻ, dẫn đến có thể mất mát dữ liệu
- [ ] Không có nhược điểm gì

27

Những kịch bản nào là đúng khi di vào khu vực quan trọng 2 luồng T0, T1 nếu hệ thống sử dụng thuật toán này * (2 Điểm)



- [ ] T0, T1, T1, T0, T0, T1
- [ ] T0, T1, T0, T1, T0, T1
- [ ] T1, T1, T1, T0, T0, T0
- [ ] T1, T0, T1, T0, T1, T0

28

Những phát biểu sau đây là đúng về giao thức truyền thông điệp UPD và TCP * (2 Điểm)

- [ ] Các gói tin được gửi theo giao thức UDP có thể bị mất trên đường truyền
- [ ] TCP là một giao thức kết nối đáng tin cậy, tức là không bị mất mát gói tin trên đường truyền
- [ ] Các gói tin được gửi theo giao thức UDP không được đảm bảo nhận dược theo thứ tự đã gửi
- [ ] TCP không đảm bảo thứ tự nhận được các gói tin giống như thứ tự đã gửi

29

Trong bài toán bữa tối của Triết gia, n>1 cùng thực thi thì cần đảm bảo điều kiện đồng bộ nào * (2 Điểm)

- [ ] Để có thể đi vào khu vực quan trọng mỗi luồng (Triết gia) cần phải lấy được 2 tài nguyên bên cạnh
- [ ] Hai luồng(Triết gia) cạnh nhau có thể đi vào cùng khu vực quan trọng

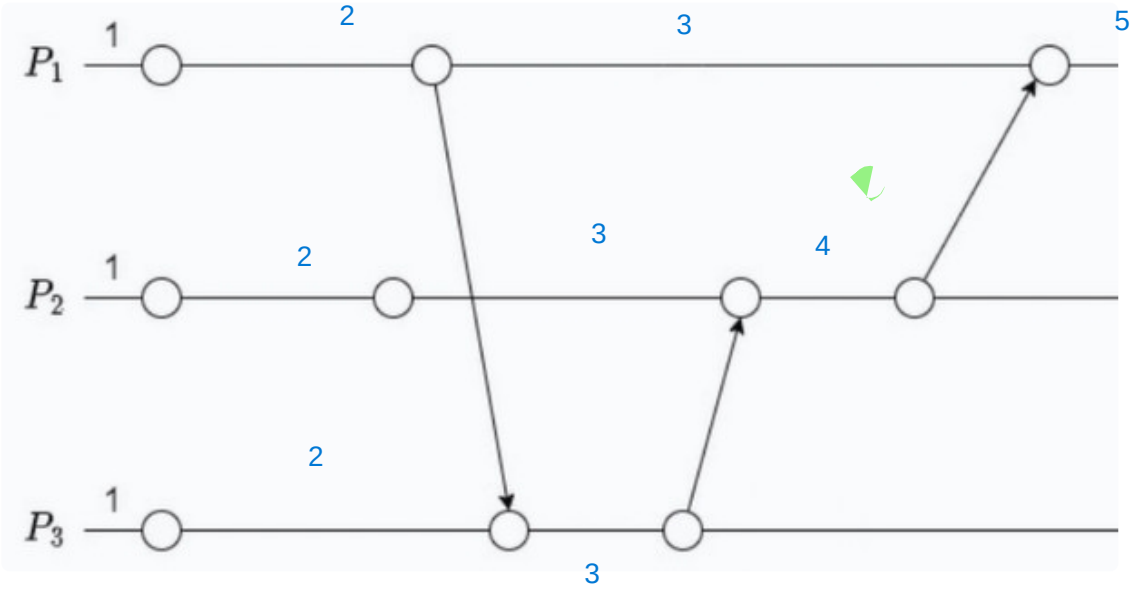
30

Những phát biểu nào sau đây là đúng về khái niệm MOM (Message-Oriented Middleware) ? * (2 Điểm)

- [ ] Trong các hệ thống phân tán được xây dựng dựa trên MOM, luôn có một thành phần trung gian, được gọi là Messaging Server, để điều phối quá trình gửi và nhận các thông điệp
- [ ] Các hệ thống phân tán được xây dựng dựa trên MOM cho phép việc truyền thông được thực hiện thông qua trao đổi bất đồng bộ các thông điệp, tức là tiến trình gửi yêu cầu sẽ không bị chặn mà có thể tiếp tục công việc của nó sau khi gửi thông điệp
- [ ] Để thực hiện được quá trình truyền thông điệp, hai tiến trình gửi và nhận bắt buộc phải cùng thực thi tại thời điểm gửi và nhận

31

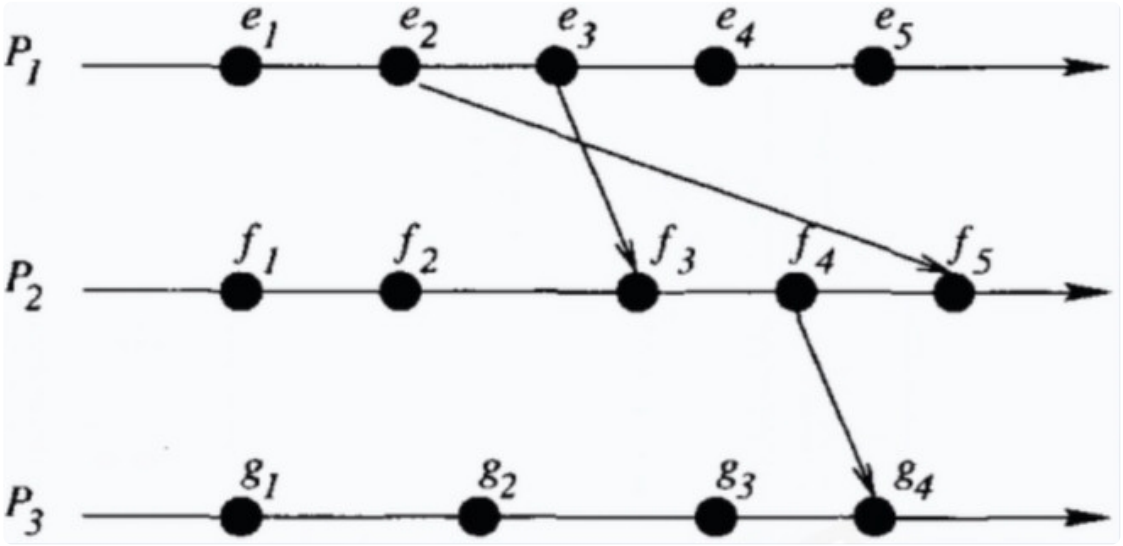
Cho sơ đồ tiến trình - thời gian, hoặc sơ đồ đã-xảy-ra-trước, như hình dưới đây.
Sử dụng **cơ chế đồng hồ logic** để đánh dấu thời gian của các trạng thái cho các tiến trình.
Dấu thời gian của trạng thái cuối cùng, sau khi sự kiện cuối cùng xảy ra, của **tiến trình P1** là gì? * (2 Điểm)



- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6

32

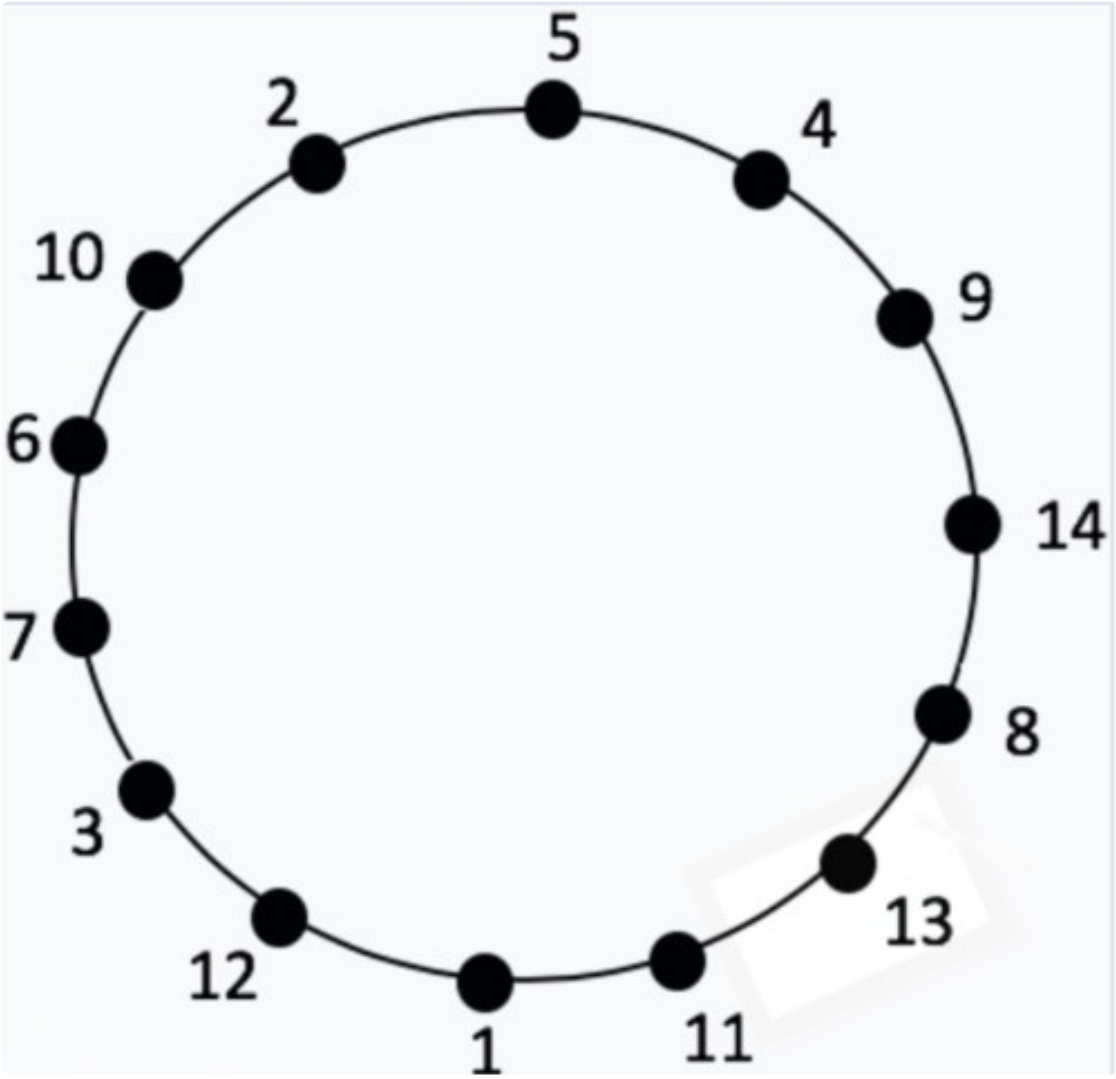
Giả sử:
- P_i là một tiến trình trong hệ thống phân tán
- e_i, f_i, g_i biểu thị các sự kiện xảy ra trên các tiến trình
Ký hiệu e → f biểu thị sự kiện e xảy ra trước sự kiện f trong ***mô hình đã-xảy-ra-trước***.
Ký hiệu e || f biểu thị sự kiện e xảy ra "*đồng thời*" với sự kiện f, tức là không có đủ thông tin để kết luận sự kiện nào xảy ra trước
Những phát biểu nào là đúng với sơ đồ tiến trình - thời gian dưới đây? *
(2 Điểm)



- [ ] e2 → g3
- [ ] f2 → g4
- [ ] f3 || g3
- [ ] e2 → f2

33

Áp dụng thuật toán bầu cử của Hirschberg-Sinclair cho sơ đồ dưới đây, tại vòng r = 0 những tiến trình nào được bầu là ứng viên cho vị trí lãnh đạo ? * (2 Điểm)



- [ ] 8 9 11 12 13 14
- [ ] 5 10 11 12 13 14
- [ ] 5 7 10 12 13 14
- [ ]

- [ ] 9 10 11 12 13 14

34

Thuật toán Peterson cho bài toán loại trừ lẫn nhau trong hệ thống đồng thời thỏa mãn các thuộc tính nào? * (2 Điểm)

- [ ] Loại trừ lẫn nhau (mutual exclusion): hai luồng bất kỳ không thể ở trong khu vực quan trọng (CS) tại cùng một thời điểm
- [ ] Tiến triển(progress): nếu một or nhiều luồng đang cố gắng để đi vào CS và không có luồng nào bên trong CS, thì ít nhất một trong các luồng sẽ thành công trong việc đi vào CS
- [ ] Có thể hoạt động được với số lượng luồng bất kỳ lớn hơn 1
- [ ] Không chết đói (starvation-freedom): nếu một luồng đang cố gắng đi vào CS, thì luồng đó cuối cùng phải đi vào CS thành công.

35

Những phát biểu nào sau đây là đúng về kỹ thuật RMI (Remote Method Invocations) trong Java? * (2 Điểm)

- [ ] Trong RMI, tiến trình khách không cần biết đến vị trí thực sự của đối tượng từ xa, i.e, không cần biết địa chỉ IP và cổng của tiến trình cung cấp dịch vụ
- [ ] Mỗi lời gọi từ xa luôn được thực hiện thông qua hai đối tượng đại diện: stub ở phía client và skeleton ở phía server
- [ ] Trong kiến trúc của RMI, ngoài hai thành phần tiến trình khách (client) và tiến trình chủ (server) - chứa đối tượng từ xa, còn có thêm thành phần thứ ba: RMIRegistry
- [ ] Hai đối tượng stub và skeleton KHÔNG bắt buộc phải được tạo ra khi thực hiện một lời gọi từ xa

36

Nhược điểm của việc sử dụng các kỹ thuật Socket hay RMI để phát triển các ứng dụng phân tán là gì? * (2 Điểm)

- [ ] Để thực hiện được quá trình truyền thông điệp với các kỹ thuật này, hai tiến trình gửi và nhận KHÔNG bắt buộc phải cùng thực thi tại thời điểm gửi và nhận đó
- [ ] Thông thường khi sử dụng các kỹ thuật này, tiến trình gửi yêu cầu sẽ bị chặn thực thi cho đến khi nhận được kết quả trả về
- [ ] Để thực hiện được quá trình truyền thông điệp, hai tiến trình gửi và nhận bắt buộc phải thực thi tại cùng một thời điểm

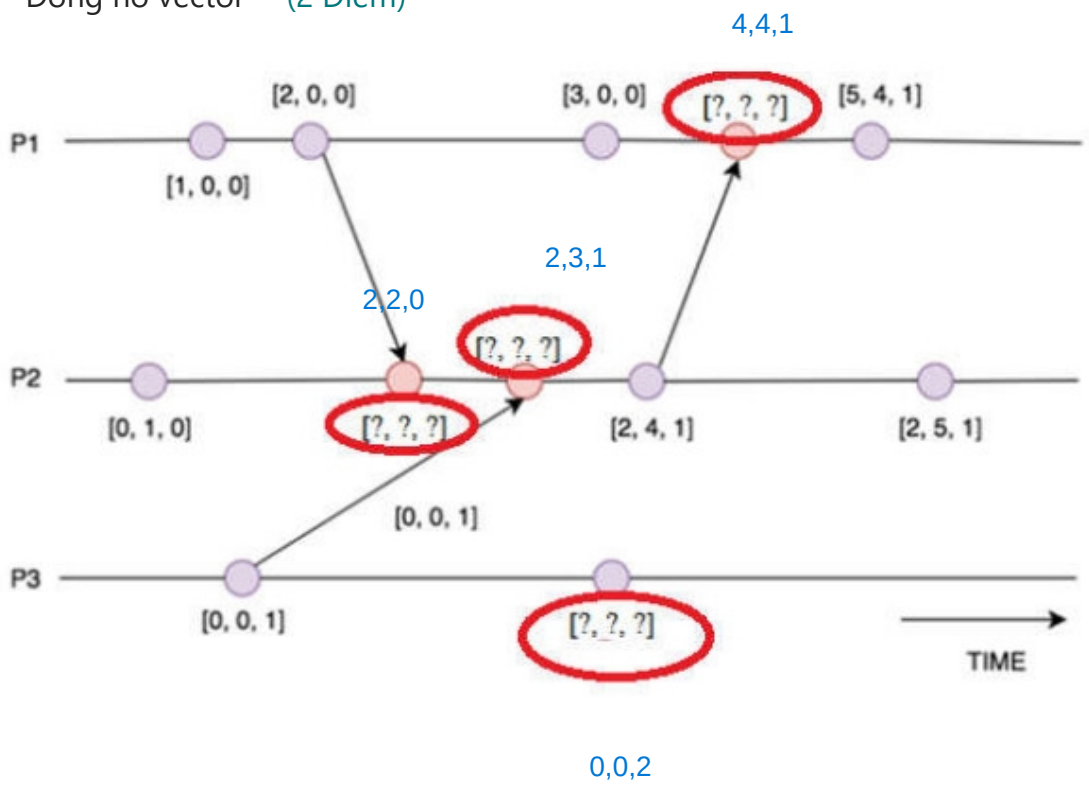
37

Trong bài toán sản xuất & tiêu thụ với 2 luồng Producer & Consumer cùng hoạt động, chúng ta cần đảm bảo những điều kiện đồng bộ nào * (2 Điểm)

- [ ] Điều kiện loại trừ lẫn nhau khi luồng Producer thực hiện đẩy dữ liệu vào bộ đệm và khi luồng Consumer thực hiện lấy dữ liệu ra khỏi bộ đệm
- [ ] Điều kiện đồng bộ cho phép 2 luồng Producer & Consumer cùng thực hiện việc đẩy và lấy dữ liệu đồng thời
- [ ] Điều kiện đồng bộ khi bộ đệm đầy, luồng Producer phải dừng lại
- [ ] Điều kiện đồng bộ khi bộ đệm rỗng, luồng Consumer phải dừng lại

38

Đồng hồ vector * (2 Điểm)



◯ Click đây

39

Những phát biểu nào sau đây là đúng về ***mô hình đã-xảy-ra-trước*** (happened-before model), được kí hiệu là →, trong một hệ thống phân tán ? * (2 Điểm)

- [ ] Trong mô hình đã-xảy-ra-trước, nếu e là sự kiện gửi của một thông điệp và f là sự kiện nhận của cùng thông điệp đó, thì e → f
- [ ] Trong mô hình đã-xảy-ra-trước, chúng ta CHỈ có được một thứ tự bộ phận trên tập các sự kiện đã xảy ra
- [ ] Trong mô hình đã-xảy-ra-trước, nếu tồn tại một sự kiện g sao cho e → g và g → f , thì e → f
- [ ] Trong mô hình đã-xảy-ra-trước, chúng ta LUÔN có được thứ tự tổng thể trên tập các sự kiện đã xảy ra, tức là với hai sự kiện bất kỳ, chúng ta luôn biết chắc chắn sự kiện nào đã xảy ra trước

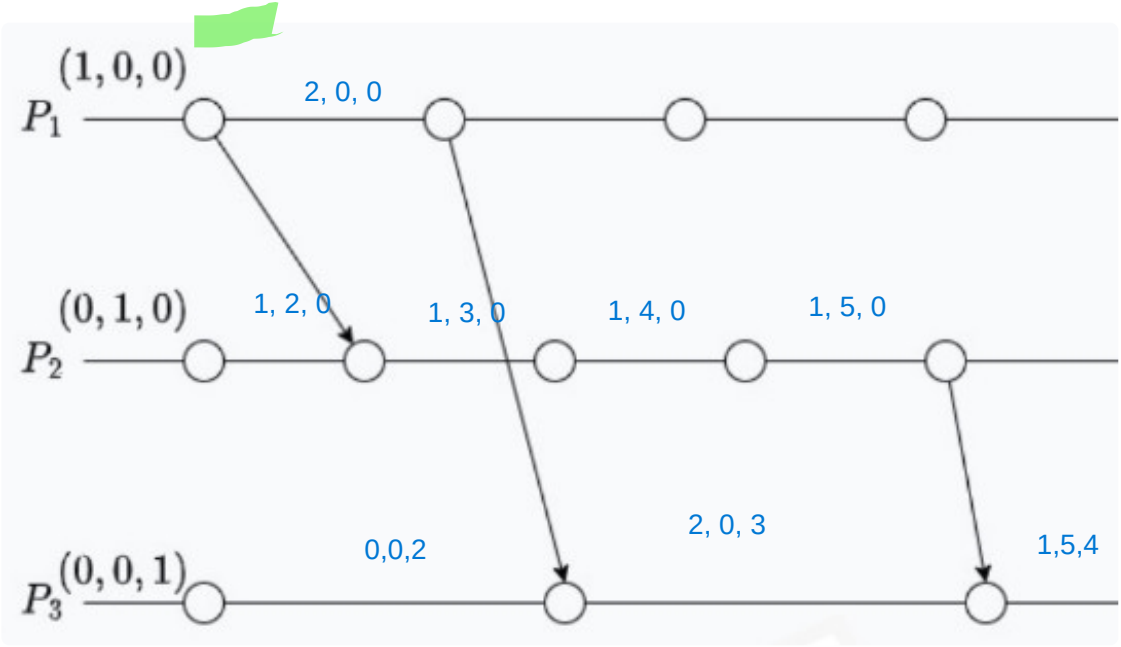
40

Nhược điểm RMI để xây dựng ứng dụng phân tán là gì * (2 Điểm)

- [ ] Để thực hiện được quá trình truyền thông điệp, hai tiến trình gửi và nhận bắt buộc phải thực thi tại cùng 1 thời điểm
- [ ] Để thực hiện được quá trình truyền thông điệp, hai tiến trình gửi và nhận không cần phải thực thi tại cùng 1 thời điểm
- [ ] Thông thường, tiến trình gửi yêu cầu sẽ bị chặn thực thi cho đến khi được nhận kết quả trả về

41

Cho sơ đồ tiến trình - thời gian, hoặc sơ đồ đã-xảy-ra-trước, như hình vẽ dưới đây.
Sử dụng **cơ chế đồng hồ vector** để đánh dấu thời gian của các trạng thái cho các tiến trình.
Dấu thời gian của trạng thái cuối cùng, sau khi sự kiện cuối cùng xảy ra, của **tiến trình P3** là gì? * (2 Điểm)



- [ ] (3,5,4)
- [ ] (2,5,4)
- [ ] (2,4,4)
- [ ] (1,5,4)

42

Trong các chương trình đồng thời, chương trình đa luồng giao tiếp với nhau bằng ? * (2 Điểm)

- [ ] Thông qua biến chia sẻ
- [ ] Thông qua truyền thông điệp, mạng truyền thông

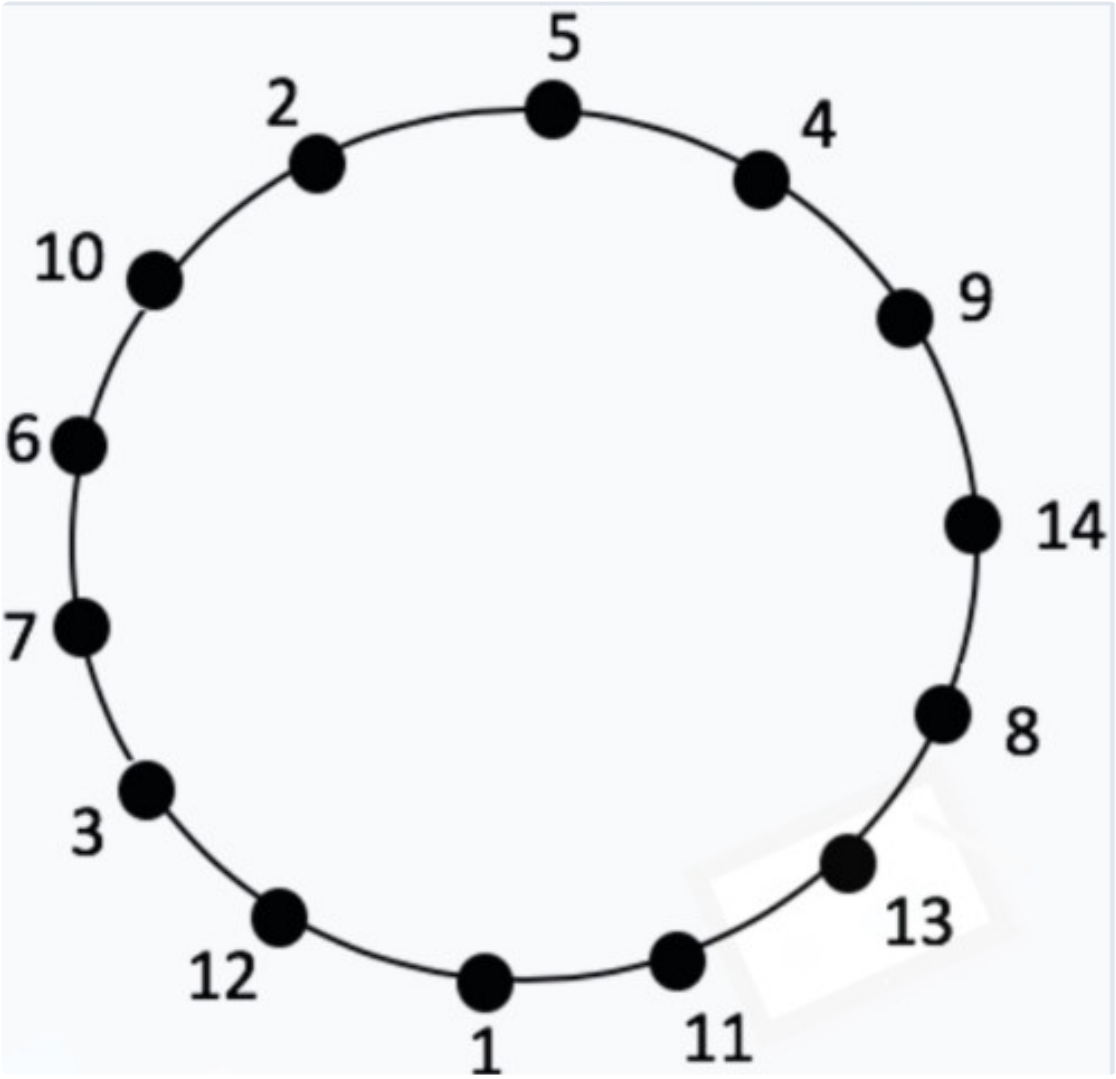
43

Giao diện Lock cho bài toán loại trừ lẫn nhau (mutex) trong một chương trình đồng thời gồm có những phương thức nào? * (2 Điểm)

- [ ] requestCS(int pid)
- [ ] release(int pid)
- [ ] enter(int pid)
- [ ] exitCS(int pid)

44

Áp dụng thuật toán bầu cử của Hirschberg-Sinclair cho sơ đồ dưới đây, tại vòng r = 1 những tiến trình nào được bầu là ứng viên cho vị trí lãnh đạo ? * (2 Điểm)

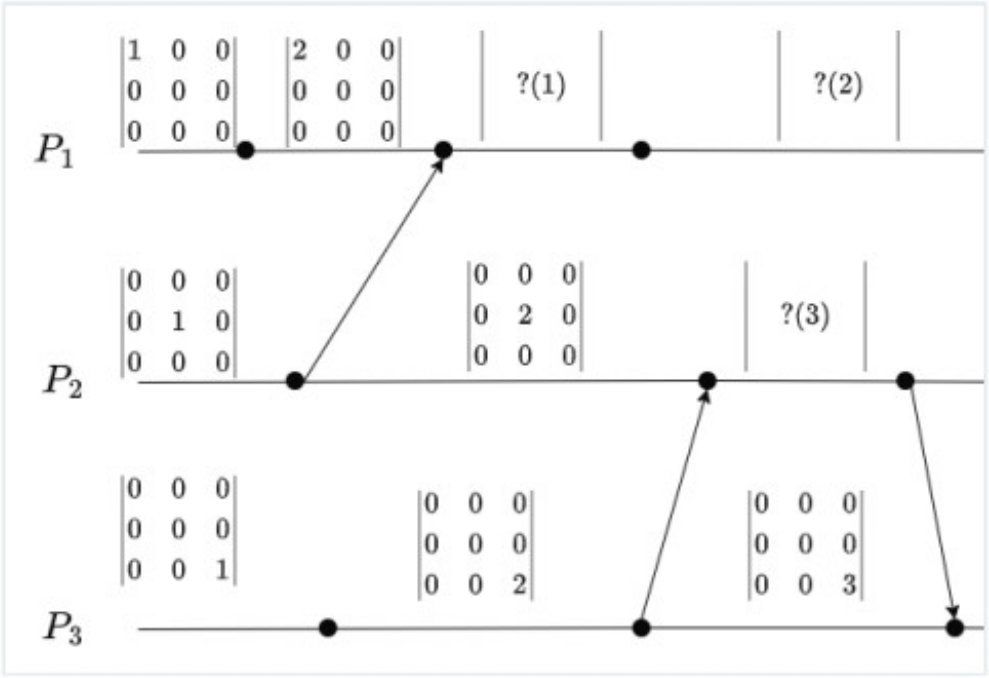


- [ ] 12 13 14
- [ ] 11 13 14
- [ ] 10 12 14
- [ ]

10 13 14

45

Tính (1), (2), (3) * (2 Điểm)



- [x] $(1) = \begin{vmatrix} 3 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}$; $(2) = \begin{vmatrix} 4 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}$; $(3) = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 3 & 2 \\ 0 & 0 & 2 \end{vmatrix}$; $(4) = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 3 & 2 \\ 0 & 3 & 4 \end{vmatrix}$

- [ ] Do Tâm mình thôi

46

Chọn đáp án * (2 Điểm)

Những phát biểu nào sau đây là đúng về Thuật toán của Lamport cho bài toán truy cập tài nguyên chia sẻ trong một hệ thống phân tán? * (1 Điểm)

- [x] Một tiến trình P_i nhận thấy nó có thể đi vào CS nếu: $\forall j \neq i: (q[i],i) < (q[j],j) \wedge (q[i],i) < (v[j],j)$, trong đó q[i], q[j]: dấu thời gian của yêu cầu đi vào CS của hai tiến trình P_i, P_j và v[j] là dấu thời gian của thông điệp ack từ tiến trình P_j được ghi nhận ở tiến trình P_i
- [x] Thuật toán sử dụng 3*(N-1) thông điệp cho mỗi lần truy cập khu vực quan trọng, với N là số lượng tiến trình, bao gồm: N - 1 thông điệp request, N - 1 thông điệp ack, N - 1 thông điệp release
- [ ] Thuật toán KHÔNG thỏa mãn thuộc tính sự sống (liveness), tức là mỗi yêu cầu đi vào khu vực quan trọng cuối cùng phải được cấp quyền để đi vào khu vực quan trọng
- [x] Thuật toán đảm bảo rằng các tiến trình đi vào khu vực quan trọng theo thứ tự dấu thời gian của yêu cầu

- [ ] Click đây

47

Semaphore đếm khác Semaphore nhị phân * (2 Điểm)

- [ ] Không có thao tác P() trong Semaphore đếm
- [ ] Không có thao tác V() trong Semaphore đếm
- [ ] Không có hàng đợi, luồng khi bị khóa trong Semaphore đếm
- [ ] Biến value trong Semaphore đếm có kiểu int, không phải kiểu boolean như trong Semaphore nhị phân
- [ ] Biến value trong Semaphore đếm có kiểu boolean, không phải kiểu int như trong Semaphore nhị phân

48

Những phát biểu nào sau đây là đúng khi khái quát về các hệ thống phân tán? * (2 Điểm)

- [ ] LUÔN tồn tại các biến chia sẻ giữa các tiến trình trong một hệ thống phân tán
- [ ] Các tiến trình trong một hệ thống phân tán giao tiếp với nhau bằng cách gửi và nhận thông điệp qua mạng truyền thông
- [ ] Mỗi tiến trình trong hệ thống phân tán luôn CHỈ gồm có một luồng
- [ ] Các tiến trình trong một hệ thống phân tán có thể KHÔNG biết được trạng thái toàn cục của hệ thống tại bất kỳ thời điểm nào
- [ ] Không tồn tại biến chia sẻ giữa các tiến trình hệ thống phân tán

49

Những phát biểu nào sau đây là đúng về bài toán truy cập tài nguyên chia sẻ, hay bài toán loại trừ lẫn nhau, trong hệ thống phân tấn? * (2 Điểm)

- [ ] Có thể sử dụng Monitor cho bài toán truy cập tài nguyên chia sẻ trong một hệ thống phân tán.
- [ ] Có thể sử dụng Semaphore cho bài toán truy cập tài nguyên chia sẻ trong một hệ thống phân tán
- [ ] Do không có bộ nhớ chia sẻ giữa các tiến trình trong hệ thống phân tán, nên không thể sử dụng các cấu trúc đồng bộ hóa như Semaphore hoặc Monitor cho bài toán này

50

Có bao nhiêu luồng * (2 Điểm)

```
public class HelloThread1 extends Thread{
    public void run() {
        for (int i = 0; i < 2; i++)
            System.out.println(Thread.currentThread().getName() + " - " + i );
```

```
    }

    public static void main(String[] args) throws InterruptedException {
        HelloThread1 t1 = new HelloThread1();
        HelloThread1 t2 = new HelloThread1();
        t1.start();
        t2.start();
    }
}
```

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4

51

Cho mã giả của chương trình đa luồng để giải quyết bài toán Sản xuất và Tiêu thụ, sử dụng monitor trong Java, như Hình dưới.
Câu lệnh gọi phương thức **notify**() của đối tượng **sharedBuffer** trong phương thức **fetch**(), được dùng với mục đích gì?
* (2 Điểm)

object sharedBuffer;

Producer	Consumer

void **deposit**() {
synchronized (sharedBuffer) {
while (bộ đệm đầy)
sharedBuffer.wait();
<*Thêm 1 phần tử vào bộ đệm*>
if (bộ đệm không rỗng)
sharedBuffer.notify();
}
}
	void **fetch**() {
	synchronized (sharedBuffer) {
	while (bộ đệm rỗng)
	sharedBuffer.wait();
	<*Lấy 1 phần tử khỏi bộ đệm*>
	if (bộ đệm không đầy)
	sharedBuffer.notify();
	}
	}

- [ ] Đánh thức luồng Tiêu thụ (Consumer) để thực hiện tiếp công việc của nó
- [ ] Đánh thức luồng Sản xuất (Producer) để thực hiện tiếp công việc của nó
- [ ] Đánh thức cả 2 luồng Sản xuất (Producer) và Tiêu thụ (Consumer)

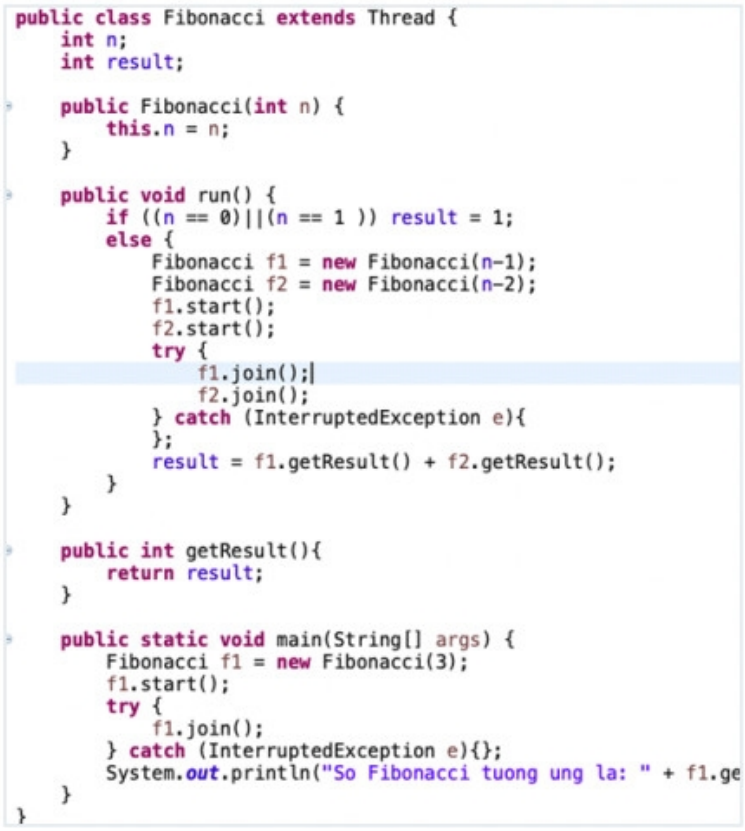
52

Những phát biểu đúng về cấu trúc đồng bộ hóa Monitor * (2 Điểm)

- [ ] Monitor hướng đối tượng tức là mỗi đối tượng mặc định sẽ đi kèm với một monitor
- [ ] Trong thời điểm chỉ có nhiều nhất một luồng chiếm giữ monitor
- [ ] Monitor không hỗ trợ khái niệm biến điều kiện

53

Có bao nhiêu luồng không tính main?
* (2 Điểm)

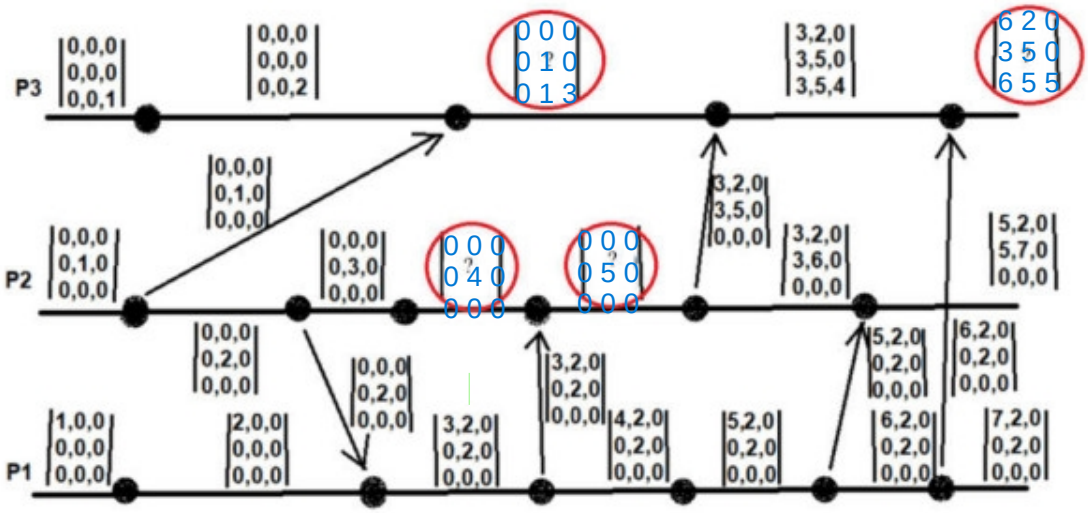


```
public class Fibonacci extends Thread {
    int n;
    int result;

    public Fibonacci(int n) {
        this.n = n;
    }

    public void run() {
        if ((n == 0)||(n == 1 )) result = 1;
        else {
            Fibonacci f1 = new Fibonacci(n-1);
            Fibonacci f2 = new Fibonacci(n-2);
            f1.start();
            f2.start();
            try {
                f1.join();
                f2.join();
            } catch (InterruptedException e){
            };
            result = f1.getResult() + f2.getResult();
        }
    }

    public int getResult(){
        return result;
    }

    public static void main(String[] args) {
        Fibonacci f1 = new Fibonacci(3);
        f1.start();
        try {
            f1.join();
        } catch (InterruptedException e){};
        System.out.println("So Fibonacci tuong ung la: " + f1.ge
    }
}
```

- ◯ 3
- ◯ 4
- ◯ 5
- ◯ 6
- ◯ 7
- ◯ 8
- ◯ 9

54

Đồng hồ ma trận * (2 Điểm)



◯ Click đây

55

Sự khác nhau giữa Monitor kiểu Hoare và Monitor kiểu Mesa là gì ?
Monitor kiểu Hoare còn được gọi là "*Signal-and-Urgent-Wait Monitor*"
Monitor kiểu Mesa còn được gọi là "*Signal-and-Continue Monitor*" *
(2 Điểm)

- [ ] Trong Monitor kiểu Mesa, không cần thiết phải có hàng đợi riêng chỉ để chứa các luồng được đánh thức. Thay vào đó, các luồng được đánh thức sẽ được chuyển sang hàng đợi dành cho các luồng mới đi vào monitor
- [ ] Trong Monitor kiểu Hoare, khi một luồng T_i đang trong monitor và gọi notify() để đánh thức luồng T_j khác, luồng T_i sẽ KHÔNG bị mất quyền chiếm giữ monitor ngay lập tức, mà nó vẫn tiếp tục công việc cho đến khi hoàn thành và ra khỏi monitor thì luồng T_j lúc này mới có thể chiếm giữ monitor
- [ ] Trong Monitor kiểu Mesa, khi một luồng T_i đang trong monitor và gọi notify() để đánh thức luồng T_j khác, luồng T_i sẽ ngay lập tức bị tạm dừng và mất quyền chiếm giữ monitor
- [ ] Send me an email receipt of my responses

56

Chương trình đồng thời khác với chương trình tuần tự ở điểm nào *
(2 Điểm)

- [ ] Chương trình tuần tự chỉ có 1 luồng thực thi, trong khi đó chương trình đồng thời có nhiều luồng cùng thực thi
- [ ] Trong chương tình đồng thời, tại một thời điểm có thể thực hiện nhiều tính toán, trong khi đó với chương trình tuần tự, tại một thời điểm có nhiều nhất 1 tính toán được thực hiện
- [ ] Hai kiểu chương trình không có sự khác biệt nào cả
- [ ] Khi chạy một chương trình đồng thời, có thể xảy ra nhiều kịch bản khác nhau, dẫn đến nhiều kết quả khác nhau

57

Xét thuật toán trong Hình dưới đây để giải quyết bài toán loại trừ lẫn nhau trong một chương trình đồng thời có 2 luồng cùng thực thi.
Những kịch bản thực thi nào sau đây của 4 câu lệnh (được đánh số màu đỏ), trong 2 phương thức **requestCS()**, sẽ khiến cho thuật toán này vi phạm điều kiện loại trừ lẫn nhau, tức là cho phép cả 2 luồng T0 và T1 đều cùng đi vào khu vực quan trọng? * (2 Điểm)

- [ ] 1->2->3->4
- [ ] 3->1->2->4
- [ ] 1->3->2->4
- [ ] 3->4->1->2
- [ ] 1->3->4->2
- [ ] 3->1->4->2

58

Cấu trúc đồng bộ hóa như Semaphore, Monitor, có thể được sử dụng để giải quyết những yêu cầu nào sau đây * (2 Điểm)

- [ ] Yêu cầu loại trừ lẫn nhau
- [ ] Yêu cầu đồng bộ

59

Thuật toán Peterson cho bài toán loại trừ lẫn nhau trong các chương trình đồng thời hoạt động đúng với bao nhiêu luồng? * (2 Điểm)

- [ ] SL bất kỳ lớn hơn 1
- [ ] Tối đa 3 luồng
- [ ] Tối đa 4 luồng
- [ ] Chính xác 2 luồng

60

Các luồng (Thread) trong một chương trình đồng thời dựa trên cơ chế khoá, chạy trên một máy tính gồm nhiều bộ vi xử lý, sử dụng mô hình nào để giao tiếp với nhau? * (2 Điểm)

- [ ] Mô hình truyền thông điệp qua mạng truyền thông
- [ ] Cả hai mô hình: Mô hình bộ nhớ chia sẻ và Mô hình truyền thông điệp qua mạng truyền thông
- [ ] Mô hình bộ nhớ chia sẻ

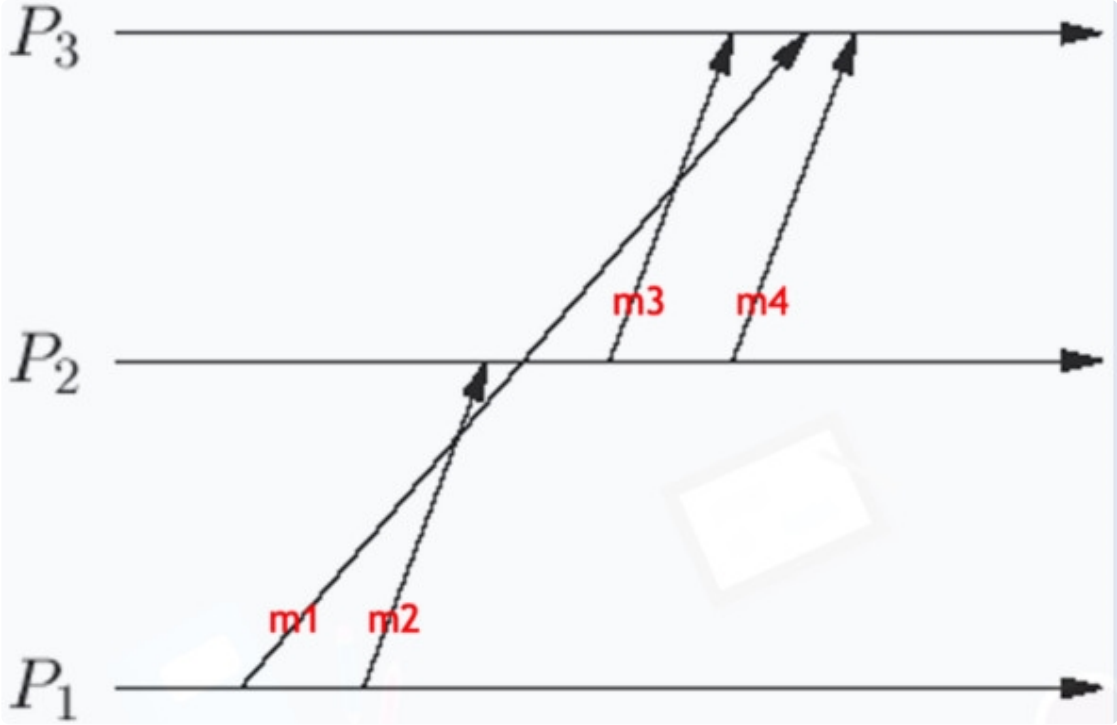
61

Những phát biểu nào đây là đúng về thuật toán Ricart & Agrawala cho bài toán truy cập tài nguyên chia sẻ trong hệ thống phân tán * (2 Điểm)

- [ ] Một tiến trình P_i nhận thấy có thể đi vào CS nếu P_i đã nhận được N-1 thông điệp okey từ N-1 tiến trình khác đồng ý cho yêu cầu đi vào CS của nó, trong đó N là số lượng tiến trình trong hệ thống phân tán
- [ ] Thuật toán sử dụng 2*(N-1) thông điệp cho mỗi lần truy cập khu vực quan trọng với N là số lượng tiến trình bao gồm N-1 thông điệp request, N - 1 thông điệp okay
- [ ] Trong thuật toán này, một tiến trình không phải lúc nào cũng gửi ngược lại thông điệp okey khi nhận được yêu cầu đi vào khu vực quan trọng của tiến trình khác
- [ ] Thuật toán không thỏa mãn thuộc tính sự sốn (liveness) tức là: mỗi yêu cầu đi vào khu vực quan trọng cuối cùng phải được cấp quyền để đi vào khu vực quan trọng

62

Cho sơ đồ tiến trình - thời gian như hình dưới.
Giả thiết P_i là một tiến trình trong một hệ thống phân tán và có 4 thông điệp (m1, m2, m3, m4) được truyền đi giữa 3 tiến trình.
Bốn thông điệp này thoả mãn kiểu thứ tự thông điệp nào sau đây? *
(2 Điểm)



- [ ] Thứ tự đồng bộ
- [ ] Thứ tự nhân quả
- [ ] Thứ tự FIFO

63

Những phát biể đúng về Socket * (2 Điểm)

- [ ] Lập trình Socket chỉ dựa trên giao thức UDP
- [ ] Socket là đối tượng được sử dụng để gửi và nhận thông điệp giữa các tiến trình trong một hệ thống phân tán
- [ ] Socket gồm 2 thành phần địa chỉ IP & Cổng
- [ ] Socket cung cấp một giao diện ở mức thấp cho việc xây dựng các chương trình phân tán

64

Cho đoạn mã giả của một chương trình đồng thời với hai luồng *t, u* như Hình dưới đây.
Giả sử các câu lệnh được thực thi một cách nguyên tử.
Sau khi hai luồng *t, u* thực thi xong các câu lệnh của mình, biến chia sẻ **counter** có thể nhận những giá trị nào sau đây ? * (2 Điểm)

int counter = 0;	
Luồng *t*	Luồng *u*
int cnt; *cnt = counter;* *for(int i = 0; i < 50; i++){* *counter = cnt + 1;* *}*	*int cnt;* *cnt = counter;* *for(int i = 0; i < 50; i++){* *counter = cnt + 1;* *}*

- [ ] counter > 100
- [ ] counter < 100
- [ ] counter = 100
- [ ] Kết quả luôn luôn bằng 100