

Họ và tên: Lương Anh Huy

Mã số sinh viên: 22520550

Lớp: IT007.O14.1

HỆ ĐIỀU HÀNH BÁO CÁO LAB 5

CHECKLIST

5.5. BÀI TẬP THỰC HÀNH

	BT 1	BT 2	BT 3	BT 4
Trình bày cách làm	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Chụp hình minh chứng	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Giải thích kết quả	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

5.6. BÀI TẬP ÔN TẬP

	BT 1
Trình bày cách làm	<input type="checkbox"/>
Chụp hình minh chứng	<input type="checkbox"/>
Giải thích kết quả	<input type="checkbox"/>

Tự chấm điểm: 9

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*

<Tên nhóm>_LAB5.pdf

5.5. BÀI TẬP THỰC HÀNH

1. Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau: $sells \leq products \leq sells + [4 \text{ số cuối của MSSV}]$

Trả lời:

- Cách làm: Sử dụng hai semaphore sem và sem1 để kiểm soát hai điều kiện là $sells \leq products$ và $products \leq sells + 550$, đặt $Condition_sell = sells + 550$

- Hình ảnh:

```
Product = 52591
Sell = 52042
Condition_Sell = 52592
Product = 52592
Sell = 52043
Condition_Sell = 52593
Product = 52593
Sell = 52044
Condition_Sell = 52594
Product = 52594
Sell = 52045
Condition_Sell = 52595
Product = 52595
Sell = 52046
Condition_Sell = 52596
Product = 52596
Sell = 52047
Condition_Sell = 52597
Product = 52597
Sell = 52048
Condition_Sell = 52598
Product = 52598
Sell = 52049
Condition_Sell = 52599
Product = 52599
Sell = 52050
Condition_Sell = 52600
Product = 52600
Sell = 52051
Condition_Sell = 52601
Product = 52601
Sell = 52052
Condition_Sell = 52602
```

Hình 1.1: Terminal của bài 1



```
1  #include <semaphore.h>
2  #include <stdio.h>
3  #include <pthread.h>
4
5  int sells = 0, products = 0;
6  sem_t sem, sem1;
7
8  void *processA()
9  {
10     while(1)
11     {
12         sem_wait(&sem);
13         sells++;
14         printf("Sell = %d\n", sells);
15         printf("Condition_Sell = %d\n", sells + 550);
16         sem_post(&sem1);
17     }
18 }
19 void *processB()
20 {
21     while(1)
22     {
23         sem_wait(&sem1);
24         products++;
25         printf("Product = %d\n", products);
26         sem_post(&sem);
27     }
28 }
29 int main()
30 {
31     sem_init(&sem, 0, 0);
32     sem_init(&sem1, 0, sells + 550);
33     pthread_t pA, pB;
34     pthread_create(&pA, NULL, &processA, NULL);
35     pthread_create(&pB, NULL, &processB, NULL);
36     while(1){}
37     return 0;
38 }
```

Hình 1.2: Code của bài 1

- Giải thích: Ta khởi tạo giá trị cho sem là 0 và sem1 có giá trị bằng sells + 550, lúc này sells đang bằng 0 nên sem1 = 550, ta chưa biết A hay B sẽ chạy trước. Giả sử A chạy trước (Trường hợp có thể dẫn đến bài toán bị sai trong 2 trường hợp), sem_wait(&sem) sẽ kiểm tra xem nếu giá trị của sem = 0 thì tiến trình A sẽ bị blocked để đảm bảo rằng sells <= products, lúc này sem đang bằng 0 nên A bị block, ta sang B, sem_wait(&sem1) cũng làm nhiệm vụ tương tự như sem_wait(&sem), với mục đích là để products <= sells + 550, lúc này sem1 vẫn đang khác 0 nên tiến trình B sẽ chạy trước, products sẽ tăng lên, sau đó sem_post(&sem) dùng để tăng giá trị của sem lên để tiến trình A được chạy và để cho sells được tăng lên. Đối với trường hợp còn lại, tức là ban đầu B chạy trước A thì điều kiện của bài toán cũng chắc chắn sẽ được thỏa mãn. Vậy ta đã giải quyết được yêu cầu của bài toán bằng việc sử dụng semaphore.

2. Cho một mảng a được khai báo như một mảng số nguyên có thể chứa n phần tử, a được khai báo như một biến toàn cục. Viết chương trình bao gồm 2 thread chạy song song:

- ✚ Một thread làm nhiệm vụ sinh ra một số nguyên ngẫu nhiên sau đó bỏ vào a. Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi thêm vào.
- ✚ Thread còn lại lấy ra một phần tử trong a (phần tử bất kỳ, phụ thuộc vào người lập trình). Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi lấy ra, nếu không có phần tử nào trong a thì xuất ra màn hình “Nothing in array a”.

Chạy thử và tìm ra lỗi khi chạy chương trình trên khi chưa được đồng bộ. Thực hiện đồng bộ hóa với semaphore.

Trả lời:

- Cách làm: Ta sẽ tạo 2 process A và B, process A dùng để tạo ngẫu nhiên giá trị và thêm vào mảng a và xuất ra số phần tử trong mảng a sau khi được thêm, process B dùng để lấy ra một phần tử bất kỳ trong mảng a và xuất ra số phần tử của mảng sau khi xóa một phần tử. Bên cạnh đó, ta sẽ thêm yêu cầu nhập số lượng phần tử tối đa của mảng a khi bắt đầu chương trình. Cuối cùng, ta sẽ kiểm tra 2 trường hợp là chưa đồng bộ và đã đồng bộ

- Hình ảnh:

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <semaphore.h>
6
7  #define MAX_SIZE 100
8  int a[MAX_SIZE];
9  int count = 0;
10 int n;
11 sem_t semA, semB;
12 pthread_mutex_t t;
13
14 void *processA()
15 {
16     while(1)
17     {
18         int randomNum = rand() % 100;
19         a[count] = randomNum;
20         count++;
21         printf("\nThem %d vao mang a", randomNum);
22         printf("\nSo phan tu co trong mang a sau khi them : %d\n", count);
23         printf("Mang a hien tai: ");
24         for (int i = 0; i < count; i++)
25         {
26             printf("%d ", a[i]);
27         }
28     }
29 }
30 void *processB()
31 {
32     while(1)
33     {
34         if (count <= 0)
35         {
36             printf("Nothing in array a\n");
37         }
38         printf("\nXoa %d ra khoi mang a", a[count - 1]);
39         count--;
40         printf("\nSo phan tu co trong mang a sau khi xoa: %d\n", count);
41         printf("Mang a hien tai: ");
42         for (int i = 0; i < count; i++)
43         {
44             printf("%d ", a[i]);
45         }
46     }
47 }
48 int main()
49 {
50     printf("Nhap so luong phan tu cua mang a: ");
51     scanf("%d", &n);
52     pthread_t pA, pB;
53     pthread_create(&pA, NULL, &processA, NULL);
54     pthread_create(&pB, NULL, &processB, NULL);
55     while(1) {}
56     return 0;
57 }
```

Hình 2.1: Code bài 2 khi chưa đồng bộ

```
Nhap so luong phan tu cua mang a: 2

Them 83 vao mang a
So phan tu co trong mang a sau khi them : 1
Mang a hien tai: 83
Them 86 vao mang a
So phan tu co trong mang a sau khi them : 2
Mang a hien tai: 83 86
Them 77 vao mang a
So phan tu co trong mang a sau khi them : 3
Mang a hien tai: 83 86 77
Them 15 vao mang a
So phan tu co trong mang a sau khi them : 4
Mang a hien tai: 83 86 77 15
Them 93 vao mang a
So phan tu co trong mang a sau khi them : 5
Mang a hien tai: 83 86 77 15 93
Them 35 vao mang a
So phan tu co trong mang a sau khi them : 6
Mang a hien tai: 83 86 77 15 93 35
```

**Hình 2.2.1: Terminal đối với trường hợp chưa đồng bộ
(Giả sử nhập $n = 3$)**

```
Xoa 86 ra khỏi mang a
So phan tu co trong mang a sau khi xoa: 1
Mang a hien tai: 83
Xoa 83 ra khỏi mang a
So phan tu co trong mang a sau khi xoa: 0
Mang a hien tai: Nothing in array a

Xoa 0 ra khỏi mang a
So phan tu co trong mang a sau khi xoa: -1
Mang a hien tai: Nothing in array a

Xoa 0 ra khỏi mang a
So phan tu co trong mang a sau khi xoa: -2
Mang a hien tai: Nothing in array a

Xoa 0 ra khỏi mang a
So phan tu co trong mang a sau khi xoa: -3
Mang a hien tai: Nothing in array a
```

**Hình 2.2.2: Terminal đối với trường hợp chưa đồng bộ
(Giả sử nhập $n = 3$)**

```
Xoa 97 ra khỏi mang a
So phan tu co trong mang a sau khi xoa: 55

So phan tu co trong mang a sau khi them : 57
Mang a hien tai: 27 90 59 63 26 40 26 72 36 11 68 67 29 82 36 32 51 37 28 75 7 74 21 99 40 42 98 13 98 90 24 90 9 81 19 36 32 55 94 4 21 42
68 28 89 72 8 58 98 36 8 53 3 67 88
Them 90 vao mang a
So phan tu co trong mang a sau khi them : 56
Mang a hien tai: 27 90 59 63 26 40 26 72 36 11 68 67 29 82 36 32 51 37 28 75 7 74 21 99 40 42 98 13 98 90 24 90 9 81 19 36 32 55 94 4 21 42
68 28 89 72 8 58 98 36 8 53 3 67 88 90
```

**Hình 2.2.3: Terminal đối với trường hợp chưa đồng bộ
(Giả sử nhập $n = 3$)**

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 #include <semaphore.h>
6
7 #define MAX_SIZE 100
8 int a[MAX_SIZE];
9 int count = 0;
10 int n;
11 sem_t semA, semB;
12 pthread_mutex_t t;
13
14 void *processA()
15 {
16     while(1)
17     {
18         sem_wait(&semB);
19         pthread_mutex_lock(&t);
20         int randomNum = rand() % 100;
21         a[count] = randomNum;
22         count++;
23         printf("\nThem %d vao mang a", randomNum);
24         printf("\nSo phan tu co trong mang a sau khi them : %d\n", count);
25         printf("Mang a hien tai: ");
26         for (int i = 0; i < count; i++)
27         {
28             printf("%d ", a[i]);
29         }
30         pthread_mutex_unlock(&t);
31         sem_post(&semA);
32     }
33 }
34 void *processB()
35 {
36     while(1)
37     {
38         if (count <= 0)
39         {
40             printf("Nothing in array a\n");
41         }
42         sem_wait(&semA);
43         pthread_mutex_lock(&t);
44         printf("\nXoa %d ra khoi mang a", a[count - 1]);
45         count--;
46         printf("\nSo phan tu co trong mang a sau khi xoa: %d\n", count);
47         printf("Mang a hien tai: ");
48         for (int i = 0; i < count; i++)
49         {
50             printf("%d ", a[i]);
51         }
52         pthread_mutex_unlock(&t);
53         sem_post(&semB);
54     }
55 }
56 int main()
57 {
58     printf("Nhap so luong phan tu cua mang a: ");
59     scanf("%d", &n);
60     sem_init(&semA, 0, 0);
61     sem_init(&semB, 0, n);
62     pthread_mutex_init(&t, NULL);
63     pthread_t pA, pB;
64     pthread_create(&pA, NULL, &processA, NULL);
65     pthread_create(&pB, NULL, &processB, NULL);
66     while(1) {}
67     return 0;
68 }
```

Hình 2.3: Code bài 2 khi đã đồng bộ

```
Them 89 vào mảng a
Số phần tử có trong mảng a sau khi thêm : 1
Mảng a hiện tại: 89
Them 5 vào mảng a
Số phần tử có trong mảng a sau khi thêm : 2
Mảng a hiện tại: 89 5
Them 17 vào mảng a
Số phần tử có trong mảng a sau khi thêm : 3
Mảng a hiện tại: 89 5 17
Them 59 vào mảng a
Số phần tử có trong mảng a sau khi thêm : 4
Mảng a hiện tại: 89 5 17 59
Them 86 vào mảng a
Số phần tử có trong mảng a sau khi thêm : 5
Mảng a hiện tại: 89 5 17 59 86
Xóa 86 ra khỏi mảng a
Số phần tử có trong mảng a sau khi xóa: 4
Mảng a hiện tại: 89 5 17 59
Xóa 59 ra khỏi mảng a
Số phần tử có trong mảng a sau khi xóa: 3
Mảng a hiện tại: 89 5 17
Xóa 17 ra khỏi mảng a
Số phần tử có trong mảng a sau khi xóa: 2
Mảng a hiện tại: 89 5
Xóa 5 ra khỏi mảng a
Số phần tử có trong mảng a sau khi xóa: 1
Mảng a hiện tại: 89
Xóa 89 ra khỏi mảng a
Số phần tử có trong mảng a sau khi xóa: 0
Mảng a hiện tại: Nothing in array a
```

Hình 2.4: Terminal đối với trường hợp đã đồng bộ
(Giả sử nhập $n = 5$)

- Giải thích:

+ Đối với trường hợp chưa đồng bộ: Dựa vào terminal, ta thấy có 3 vấn đề xảy ra

*Vấn đề 1: Số lượng phần tử được thêm vào mảng vượt xa so với n phần tử mà ta đã nhập vào và không biết khi nào việc thêm phần tử sẽ dừng lại

*Vấn đề 2: Số lượng phần tử được xóa khỏi mảng không logic, giả sử mảng không có phần tử nào thì khi ta lấy một phần tử ra khỏi mảng thì số phần tử trong mảng vẫn là 0, tuy nhiên số lượng phần tử trong mảng lúc này đã thành số âm và cứ giảm dần mà không biết điểm dừng, tương tự như vấn đề 1

*Vấn đề 3: Số lượng phần tử không nhất quán. Ở hình 2.2.3 ta thấy rằng sau khi xóa một phần tử lúc này chương trình xuất ra số lượng phần tử hiện đang là 55, sau đó ta

chưa thấy phần tử nào được thêm vào mảng nhưng chương trình lại xuất ra số phần tử sau khi thêm là 57, và tiếp theo thì sau khi thêm một phần tử thì số lượng phần tử lại là 56, điều này hoàn toàn không hợp lý, vậy vấn đề ở đây là do cả 2 tiến trình A và B đều đi vào vùng tranh chấp nên dữ liệu không được nhất quán

+ Đối với trường hợp đã đồng bộ: Giải pháp lúc này để khắc phục các vấn đề trên là sử dụng semaphore để kiểm soát việc tăng giảm của biến count, đảm bảo sự nhất quán của dữ liệu số lượng phần tử và sử dụng mutex để đảm bảo rằng chỉ một trong hai tiến trình thực thi tại một thời điểm, đảm bảo dữ liệu. Ta sẽ khởi tạo semB có giá trị ứng với số lượng phần tử tối đa n nhập vào và semA có giá trị là 0. Ở tiến trình A, ta kiểm tra sem_wait(&semB) để ràng buộc tiến trình A không được thêm vào mảng quá n phần tử như vấn đề 1. Ở tiến trình B, ta kiểm tra sem_wait(&semA) để ràng buộc tiến trình B không được giảm số lượng phần tử khi trong mảng chưa có phần tử nào để khắc phục vấn đề thứ 2. Và ta sử dụng pthread_mutex_lock và pthread_mutex_unlock để kiểm soát vùng tranh chấp, đối với bài toán này vùng tranh chấp ở tiến trình A là đoạn code khi thêm và xuất ra số lượng phần tử, ở tiến trình B là đoạn code khi xóa và xuất ra số lượng phần tử nên ta sẽ đặt lock và unlock giới hạn 2 đầu của 2 đoạn code này để đảm bảo rằng chỉ có một tiến trình tiến vào vùng tranh chấp để xử lý vấn đề 3. Vậy là ta đã hoàn tất việc tìm ra lỗi và khắc phục đối với 2 trường hợp là chưa đồng bộ và đã đồng bộ.

3. Cho 2 process A và B chạy song song như sau:

int x = 0;	
PROCESS A	PROCESS B
processA() { while(1){ x = x + 1; if (x == 20) x = 0; print(x); }	processB() { while(1){ x = x + 1; if (x == 20) x = 0; print(x); }

}	}
}	}

Hiện thực mô hình trên C trong hệ điều hành Linux và nhận xét kết quả.

Trả lời:

- Cách làm: Hiện thực mô hình trên vào ngôn ngữ C
- Hình ảnh:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE
Executed by process A| x = 13
Executed by process A| x = 14
Executed by process A| x = 15
Executed by process A| x = 16
Executed by process A| x = 17
Executed by process A| x = 18
Executed by process A| x = 19
Executed by process A| x = 0
Executed by process A| x = 1
Executed by process A| x = 2
Executed by process A| x = 3
Executed by process A| x = 4
Executed by process A| x = 5
Executed by process A| x = 6
Executed by process A| x = 7
Executed by process A| x = 8
Executed by process A| x = 9
Executed by process A| x = 10
Executed by process B| x = 18
Executed by process B| x = 12
Executed by process B| x = 13
Executed by process B| x = 14
Executed by process B| x = 15
```

Hình 3.1: Terminal của bài 3

```
1  #include <semaphore.h>
2  #include <stdio.h>
3  #include <pthread.h>
4
5  int x = 0;
6  void *processA()
7  {
8      while(1)
9      {
10         x += 1;
11         if (x == 20)
12             x = 0;
13         printf("Executed by process A| x = %d\n", x);
14     }
15 }
16 void *processB()
17 {
18     while(1)
19     {
20         x += 1;
21         if (x == 20)
22             x = 0;
23         printf("Executed by process B| x = %d\n", x);
24     }
25 }
26 int main()
27 {
28     pthread_t pA, pB;
29     pthread_create(&pA, NULL, &processA, NULL);
30     pthread_create(&pB, NULL, &processB, NULL);
31     while (1) {}
32     return 0;
33 }
```

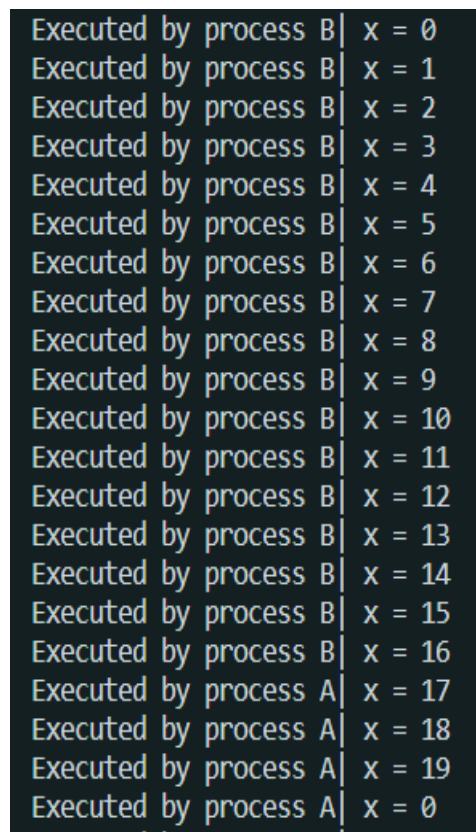
Hình 3.2: Code của bài 3

Giải thích: Vì 2 tiến trình A và B chạy song song với nhau nên cả A và B đều sẽ tiến vào vùng tranh chấp để thay đổi giá trị của x, khi đó lỗi đồng bộ sẽ xảy ra, dựa trên terminal ta thấy được rằng tiến trình A đang được thi thực thi và $x = 10$, sau đó tiến trình B thực thi và x lần lượt bằng 18 và 12 trong khi điều kiện là x chỉ tăng 1 đơn vị, vì thế nên lỗi ở đây là việc dữ liệu x không được nhất quán giữa 2 tiến trình

4. Đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình Bài 3.

Cách làm: Sử dụng mutex để khắc phục lỗi đồng bộ của bài 3

Hình ảnh:



Executed by process B	x = 0
Executed by process B	x = 1
Executed by process B	x = 2
Executed by process B	x = 3
Executed by process B	x = 4
Executed by process B	x = 5
Executed by process B	x = 6
Executed by process B	x = 7
Executed by process B	x = 8
Executed by process B	x = 9
Executed by process B	x = 10
Executed by process B	x = 11
Executed by process B	x = 12
Executed by process B	x = 13
Executed by process B	x = 14
Executed by process B	x = 15
Executed by process B	x = 16
Executed by process A	x = 17
Executed by process A	x = 18
Executed by process A	x = 19
Executed by process A	x = 0

Hình 4.1: Terminal của bài 4

```
1  #include <stdio.h>
2  #include <semaphore.h>
3  #include <pthread.h>
4
5  int x = 0;
6  pthread_mutex_t t;
7  void *processA()
8  {
9      while(1)
10     {
11         pthread_mutex_lock(&t);
12         x += 1;
13         if (x == 20)
14             x = 0;
15         printf("Executed by process A| x = %d\n", x);
16         pthread_mutex_unlock(&t);
17     }
18 }
19 void *processB()
20 {
21     while(1)
22     {
23         pthread_mutex_lock(&t);
24         x += 1;
25         if (x == 20)
26             x = 0;
27         printf("Executed by process B| x = %d\n", x);
28         pthread_mutex_unlock(&t);
29     }
30 }
31 int main()
32 {
33     pthread_t pA, pB;
34     pthread_mutex_init(&t, NULL);
35     pthread_create(&pA, NULL, &processA, NULL);
36     pthread_create(&pB, NULL, &processB, NULL);
37     while (1) {}
38     return 0;
39 }
```

Hình 4.2: Code của bài 4

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

Giải thích: Ta sẽ sử dụng mutex để khắc phục lỗi xảy ra ở bài 3. Vì vùng tranh chấp nằm trong vòng lặp while ở hai tiến trình A và B nên ta đặt 2 dòng lệnh `pthread_mutex_lock` và `pthread_mutex_unlock` ở đầu và cuối 2 đoạn code trong vòng lặp while để đảm bảo vùng tranh chấp rằng trong một thời điểm chỉ có một tiến trình được tiến vào vùng tranh chấp, khi này giá trị của x sẽ được đồng bộ, và các giá trị của x sẽ hợp lý, thỏa mãn điều kiện của đề bài

5.6. BÀI TẬP ÔN TẬP

1. Biến ans được tính từ các biến x1, x2, x3, x4, x5, x6 như sau:

$$w = x1 * x2; (a)$$

$$v = x3 * x4; (b)$$

$$y = v * x5; (c)$$

$$z = v * x6; (d)$$

$$y = w * y; (e)$$

$$z = w * z; (f)$$

$$ans = y + z; (g)$$

Giả sử các lệnh từ (a) \rightarrow (g) nằm trên các thread chạy song song với nhau. Hãy lập trình mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự sau:

- 🚦 (c), (d) chỉ được thực hiện sau khi v được tính
- 🚦 (e) chỉ được thực hiện sau khi w và y được tính
- 🚦 (g) chỉ được thực hiện sau khi y và z được tính

Trả lời...