

Họ và tên: Nguyễn Trần Bảo Anh

Mã số sinh viên: 22520066

Lớp: IT007.O21.CNVN.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 type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chụp hình minh chứng	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Giải thích kết quả	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input 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

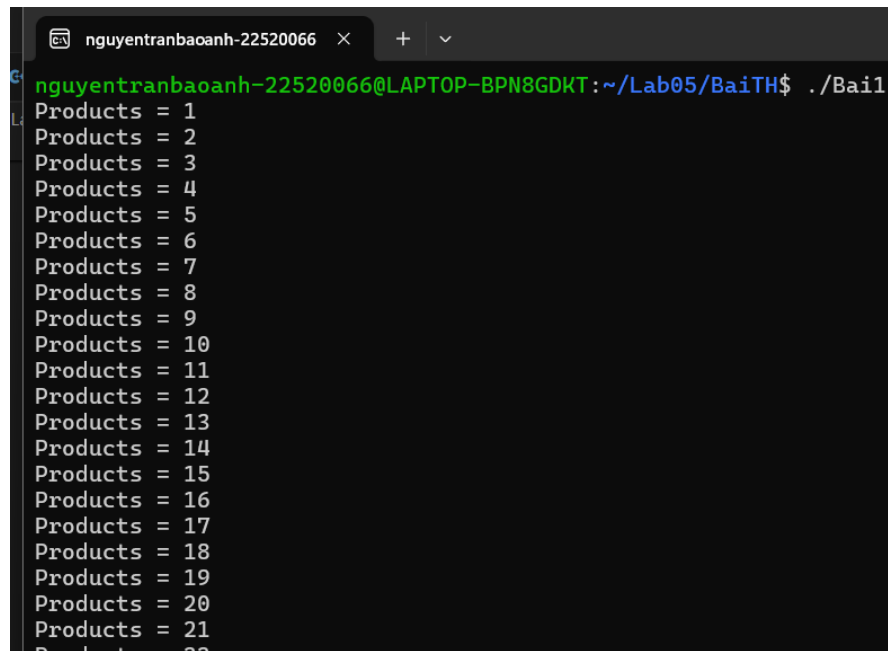
5.5. BÀI TẬP THỰC HÀNH

1. Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau:
sells <= products <= sells + [4 số cuối của MSSV]

Trình bày cách làm

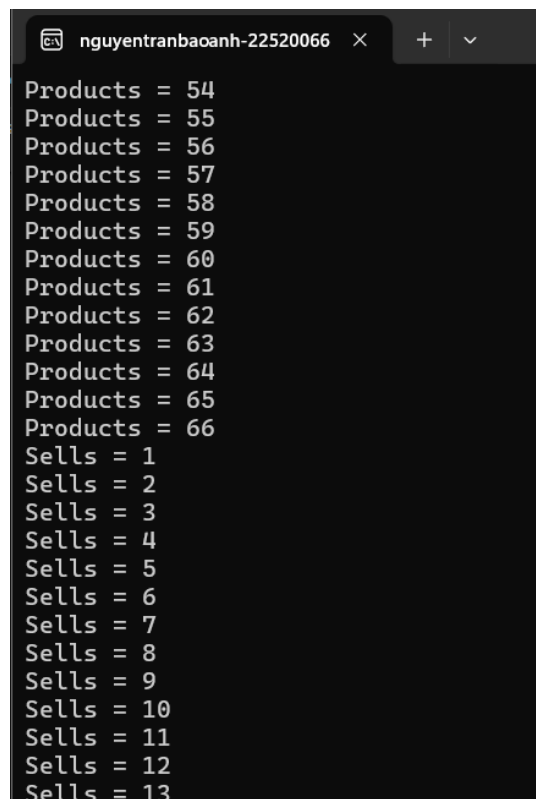
```
Lab05 > BaiTH > C Bai1.c > main()
1  #include <stdio.h>
2  #include <semaphore.h>
3  #include <pthread.h>
4
5  int sells = 0, products = 0;
6  sem_t sem1, sem2;
7
8  void *ProcessA(void* mess)
9  {
10     while(1)
11     {
12         sem_wait(&sem1);
13         sells++;
14         printf("Sells = %d\n", sells);
15         sem_post(&sem2);
16     }
17 }
18
19 void *ProcessB(void* mess)
20 {
21     while(1)
22     {
23         sem_wait(&sem2);
24         products++;
25         printf("Products = %d\n", products);
26         sem_post(&sem1);
27     }
28 }
29
30 int main()
31 {
32     sem_init(&sem1, 0, 0);
33     sem_init(&sem2, 0, 66);
34     pthread_t pA, pB;
35     pthread_create(&pA, NULL, &ProcessA, NULL);
36     pthread_create(&pB, NULL, &ProcessB, NULL);
37     while(1){}
38
39     return 0;
40 }
```

Chụp hình minh chứng



```
nguyentranbaoanh-22520066 x + v
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab05/BaiTH$ ./Bai1
Products = 1
Products = 2
Products = 3
Products = 4
Products = 5
Products = 6
Products = 7
Products = 8
Products = 9
Products = 10
Products = 11
Products = 12
Products = 13
Products = 14
Products = 15
Products = 16
Products = 17
Products = 18
Products = 19
Products = 20
Products = 21
Products = 22
```

Hình 1: Process thực hiện việc cộng Products được chạy trước.



```
nguyentranbaoanh-22520066 x + v
Products = 54
Products = 55
Products = 56
Products = 57
Products = 58
Products = 59
Products = 60
Products = 61
Products = 62
Products = 63
Products = 64
Products = 65
Products = 66
Sells = 1
Sells = 2
Sells = 3
Sells = 4
Sells = 5
Sells = 6
Sells = 7
Sells = 8
Sells = 9
Sells = 10
Sells = 11
Sells = 12
Sells = 13
```

Hình 2: Sau khi đạt giới hạn, Products bị buộc dừng bởi sem_wait và Sells được chạy.

Giải thích kết quả

sem1 đóng vai trò là điều kiện ($sells \leq products$), sem2 đóng vai trò là điều kiện ($products \leq sells + 66$). Khi chương trình được nạp, giả sử ProcessA được nạp trước. Khi đó A sẽ bị khóa lại với bởi biến sem1 (khởi tạo = 0). Vì vậy ProcessB sẽ được chạy. Products và sem1 (bởi hàm sem_post) sẽ được tăng lên cho đến khi sem2=0 (được giảm bởi hàm sem_wait). Khi đó ProcessA sẽ được chạy. Sells và sem2 (bởi hàm sem_post) sẽ tăng cho đến khi sem1=0 (được giảm bởi hàm sem_wait). Hai tiến trình này sẽ được lặp đi lặp lại trong 1 vòng lặp vô hạn.

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”.

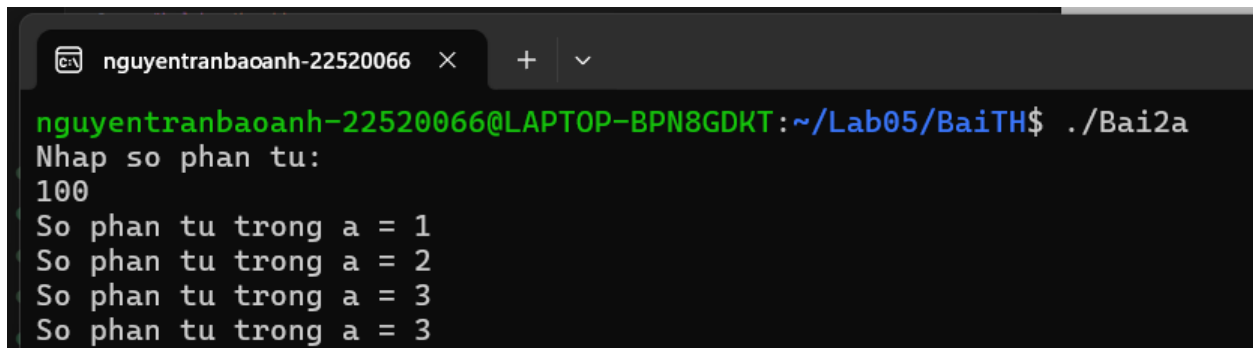
Chạy thử và tìm ra lỗi khi chạy chương trình trên khi chưa được đồng bộ. Thực hiện đồng bộ hóa với semaphore.

a. Chưa đồng bộ

Trình bày cách làm

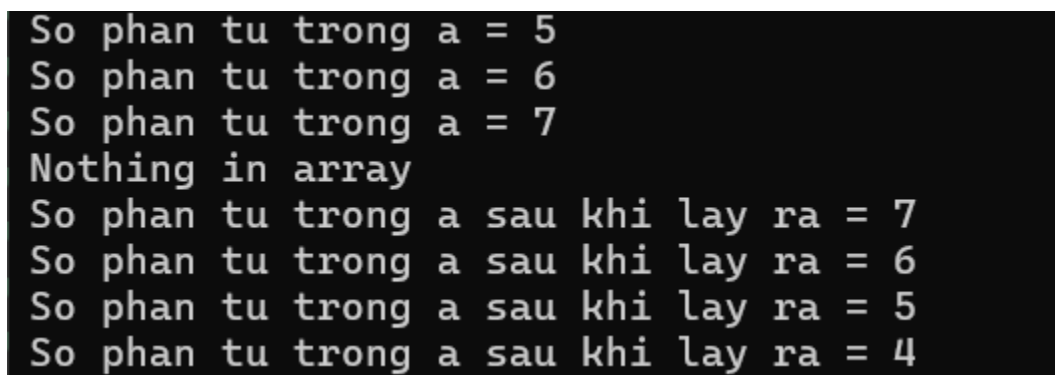
```
1 // Chưa đồng bộ
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <pthread.h>
6 #include <time.h>
7
8 int* a;
9 int n;
10 int iNum = 0;
11
12 void Arrange(int *a, int x)
13 {
14     if(x == iNum)
15         iNum--;
16     else
17     {
18         for(int i = x; i < iNum-1; i++)
19             a[i] = a[i+1];
20         iNum--;
21     }
22 }
23
24 void* ProcessA(void* mess)
25 {
26     while(1)
27     {
28         srand((int)time(0));
29         a[iNum] = rand();
30         iNum++;
31         printf("So phan tu trong a = %d\n", iNum);
32     }
33 }
34
35 void* ProcessB(void* mess)
36 {
37     while(1)
38     {
39         srand((int)time(0));
40         if(iNum == 0)
41             printf("Nothing in array\n");
42         else
43         {
44             int r = rand() % iNum;
45             Arrange(a, r);
46             printf("So phan tu trong a sau khi lay ra = %d\n", iNum);
47         }
48     }
49 }
50
51 int main()
52 {
53     printf("Nhap so phan tu: \n");
54     scanf("%d", &n);
55     a = (int*)malloc(n*sizeof(int));
56     pthread_t pA, pB;
57     pthread_create(&pA, NULL, &ProcessA, NULL);
58     pthread_create(&pB, NULL, &ProcessB, NULL);
59     while(1){}
60
61     return 0;
62 }
```

Chụp hình minh chứng



```
nguyentranbaoanh-22520066 x + v
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab05/BaiTH$ ./Bai2a
Nhap so phan tu:
100
So phan tu trong a = 1
So phan tu trong a = 2
So phan tu trong a = 3
So phan tu trong a = 3
```

Hình 3: Tiến trình sinh ra phần tử và bỏ vào a được chạy trước.



```
So phan tu trong a = 5
So phan tu trong a = 6
So phan tu trong a = 7
Nothing in array
So phan tu trong a sau khi lay ra = 7
So phan tu trong a sau khi lay ra = 6
So phan tu trong a sau khi lay ra = 5
So phan tu trong a sau khi lay ra = 4
```

Hình 4: Lỗi xuất hiện khi chưa đồng bộ.

b. Đã đồng bộ

Trình bày cách làm

```
3 // Chưa đồng bộ
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <pthread.h>
8 #include <time.h>
9 #include <semaphore.h>
10
11 int* a;
12 int n;
13 int iNum = 0;
14 sem_t sem1, sem2, busy;
15
16 void Arrange(int *a, int x)
17 {
18     if(x == iNum)
19         iNum--;
20     else
21     {
22         for(int i = x; i < iNum-1; i++)
23             a[i] = a[i+1];
24         iNum--;
25     }
26 }
27
28 void* ProcessA(void* mess)
29 {
30     while(1)
31     {
32         sem_wait(&sem2);
33         sem_wait(&busy);
34         srand((int)time(0));
35         a[iNum] = rand();
36         iNum++;
37         printf("Số phần tử trong a = %d\n", iNum);
38         sem_post(&sem1);
39         sem_post(&busy);
40     }
41 }
42
43 void* ProcessB(void* mess)
44 {
45     while(1)
46     {
47         sem_wait(&sem1);
48         sem_wait(&busy);
49         srand((int)time(0));
50         if(iNum == 0)
51             printf("Nothing in array\n");
52         else
53         {
54             int r = rand() % iNum;
55             Arrange(a, r);
56             printf("Số phần tử trong a sau khi lấy ra = %d\n", iNum);
57         }
58         sem_post(&sem2);
59         sem_post(&busy);
60     }
61 }
62
63 int main()
64 {
65     printf("Nhập số phần tử: \n");
66     scanf("%d", &n);
67     a = (int*)malloc(n*sizeof(int));
68     sem_init(&sem1, 0, 0);
69     sem_init(&sem2, 0, n);
70     sem_init(&busy, 0, 1);
71     pthread_t pA, pB;
72     pthread_create(&pA, NULL, &ProcessA, NULL);
73     pthread_create(&pB, NULL, &ProcessB, NULL);
74     while(1){}
75
76     return 0;
77 }
```

Chụp hình minh chứng

```
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab05/BaiTH$ ./Bai2b
Nhap so phan tu:
10
So phan tu trong a = 1
So phan tu trong a = 2
So phan tu trong a = 3
So phan tu trong a sau khi lay ra = 2
So phan tu trong a sau khi lay ra = 1
So phan tu trong a sau khi lay ra = 0
```

Hình 5: Số phần tử trong a được thêm vào và lấy ra luôn có sự đồng nhất về dữ liệu.

```
So phan tu trong a = 8
So phan tu trong a = 9
So phan tu trong a = 10
So phan tu trong a sau khi lay ra = 9
So phan tu trong a sau khi lay ra = 8
So phan tu trong a sau khi lay ra = 7
So phan tu trong a sau khi lay ra = 6
So phan tu trong a sau khi lay ra = 5
So phan tu trong a sau khi lay ra = 4
So phan tu trong a sau khi lay ra = 3
So phan tu trong a sau khi lay ra = 2
So phan tu trong a sau khi lay ra = 1
So phan tu trong a sau khi lay ra = 0
So phan tu trong a = 1
So phan tu trong a = 2
So phan tu trong a = 3
So phan tu trong a = 4
So phan tu trong a = 5
So phan tu trong a = 6
So phan tu trong a = 7
So phan tu trong a = 8
So phan tu trong a = 9
So phan tu trong a = 10
So phan tu trong a sau khi lay ra = 9
```

Hình 6: Tương tự khi 2 bên đổi vị trí.

Giải thích kết quả

Hàm `arrange` đóng vai trò sắp xếp lại hàm khi lấy ngẫu nhiên một phần tử trong `a` ra. Biến `sem1` đóng vai trò điều kiện đảm bảo rằng sẽ không thể lấy phần tử trong `A` khi không có phần tử nào. Biến `sem2` đóng vai trò điều kiện đảm bảo rằng sẽ không thể thêm phần tử quá số lượng phần tử được cho phép. Khi chương trình được nạp, giả sử `ProcessB` được nạp trước sẽ bị khóa bởi biến `sem1` (khởi tạo = 0). Khi đó `ProcessA` được chạy, tiếp tục thêm vào phần tử có giá trị ngẫu nhiên vào `a` và tăng `sem1` (bởi hàm `sem_post`) cho đến khi `sem2=0` (được giảm bởi hàm `sem_wait`). Khi đó `ProcessB` được chạy, lấy một phần tử ngẫu nhiên trong `a` ra và tăng `sem2` `sem1` (bởi hàm `sem_post`) cho đến khi `sem1 = 0` (được giảm bởi hàm `sem_wait`). Hai tiến trình này sẽ được lặp đi lặp lại trong 1 vòng lặp vô hạn.

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

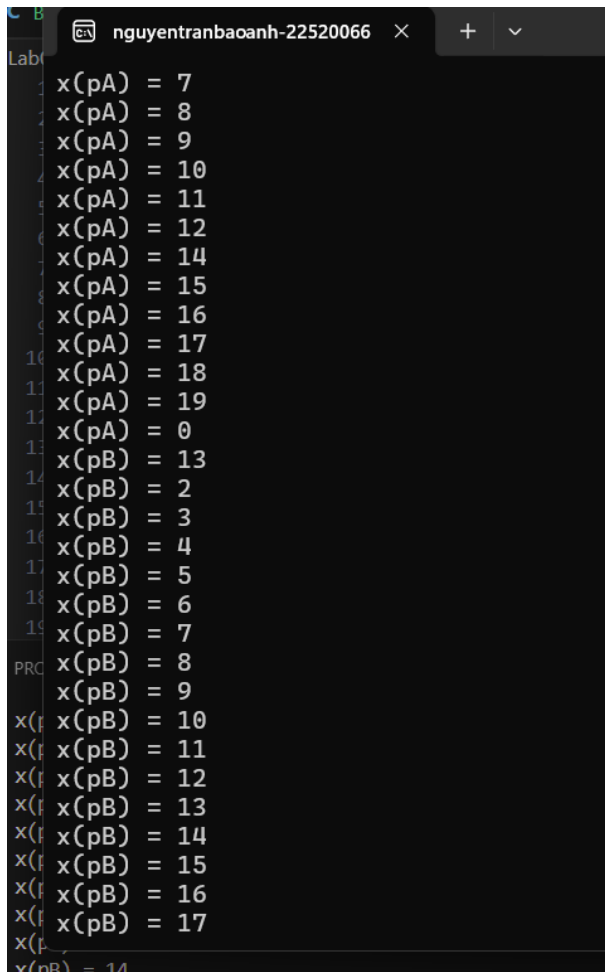
– <code>int x = 0;</code>	
– PROCESS A	– PROCESS B
– <code>processA()</code>	– <code>processB()</code>
– {	– {
– <code>while(1){</code>	– <code>while(1){</code>
– <code>x =</code>	– <code>x =</code>
– <code>x + 1;</code>	– <code>x + 1;</code>
– <code>if (x</code>	– <code>if (x</code>
– <code>== 20)</code>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

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

Trình bày cách làm

```
C Bai2a.c U      C Bai3.c U X      C Bai2b.c U
Lab05 > BaiTH > C Bai3.c > ProcessA(void *)
1  #include <stdio.h>
2  #include <semaphore.h>
3  #include <pthread.h>
4
5  int x = 0;
6
7  void* ProcessA(void* mess)
8  {
9      while(1)
10     {
11         x = x + 1;
12         if(x == 20)
13             x = 0;
14         printf("x(pA) = %d\n", x);
15     }
16 }
17
18 void* ProcessB()
19 {
20     while(1){
21         x = x + 1;
22         if (x == 20)
23             x = 0;
24         printf("x(pB) = %d\n", x);
25     }
26 }
27
28 int main()
29 {
30     pthread_t pA, pB;
31     pthread_create(&pA, NULL, &ProcessA, NULL);
32     pthread_create(&pB, NULL, &ProcessB, NULL);
33     while(1){}
34
35     return 0;
36 }
```

Chụp hình minh chứng



```
nguyentranbaoanh-22520066 x + v
Lab
x(pA) = 7
x(pA) = 8
x(pA) = 9
x(pA) = 10
x(pA) = 11
x(pA) = 12
x(pA) = 14
x(pA) = 15
x(pA) = 16
x(pA) = 17
x(pA) = 18
x(pA) = 19
x(pA) = 0
x(pB) = 13
x(pB) = 2
x(pB) = 3
x(pB) = 4
x(pB) = 5
x(pB) = 6
x(pB) = 7
x(pB) = 8
x(pB) = 9
x(pB) = 10
x(pB) = 11
x(pB) = 12
x(pB) = 13
x(pB) = 14
x(pB) = 15
x(pB) = 16
x(pB) = 17
x(pB) = 14
```

Giải thích kết quả

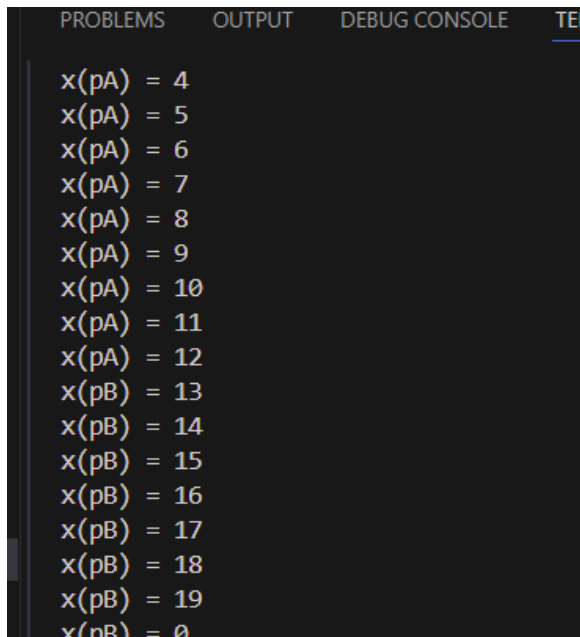
- Lỗi xuất hiện khi chưa được đồng bộ. Process B đã lấy giá trị của biến x ở thời điểm Process A đang tính toán x=12, do đó sau khi Process A kết thúc quá trình chạy thì Process B lập tức in ra kết quả đã tính trước đó.

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

Trình bày cách làm

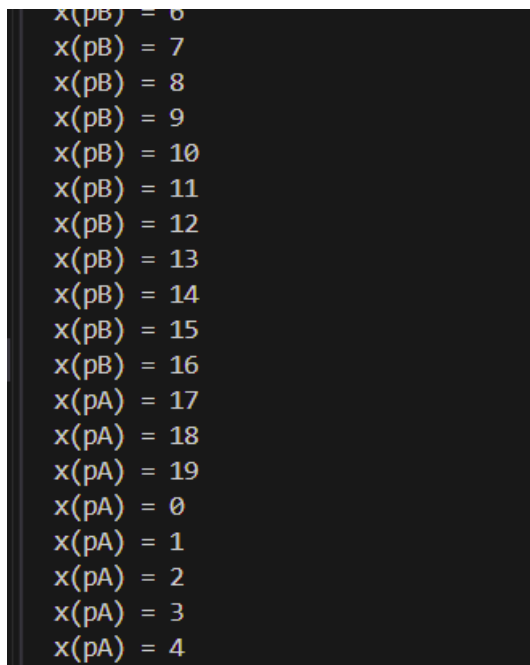
```
Lab05 > BaiTH > C Bai4.c > ProcessA(void *)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  int x = 0;
6  pthread_mutex_t mutex;
7
8  void* ProcessA(void* mess)
9  {
10     while(1)
11     {
12         pthread_mutex_lock(&mutex);
13         x = x + 1;
14         if(x == 20)
15             x = 0;
16         printf("x(pA) = %d\n", x);
17         pthread_mutex_unlock(&mutex);
18     }
19 }
20
21 void* ProcessB(void* mess)
22 {
23     while(1)
24     {
25         pthread_mutex_lock(&mutex);
26         x = x + 1;
27         if (x == 20)
28             x = 0;
29         printf("x(pB) = %d\n", x);
30         pthread_mutex_unlock(&mutex);
31     }
32 }
33
34 int main()
35 {
36     pthread_t pA, pB;
37     pthread_mutex_init(&mutex, NULL);
38     pthread_create(&pA, NULL, &ProcessA, NULL);
39     pthread_create(&pB, NULL, &ProcessB, NULL);
40     while(1){}
41
42     return 0;
43 }
```

Chụp hình minh chứng



```
PROBLEMS OUTPUT DEBUG CONSOLE TE
x(pA) = 4
x(pA) = 5
x(pA) = 6
x(pA) = 7
x(pA) = 8
x(pA) = 9
x(pA) = 10
x(pA) = 11
x(pA) = 12
x(pB) = 13
x(pB) = 14
x(pB) = 15
x(pB) = 16
x(pB) = 17
x(pB) = 18
x(pB) = 19
x(pB) = 0
```

Hình 7: Process A kết thúc và lập tức Process B tiếp tục tính toán theo kết quả cuối cùng của Process A



```
x(pB) = 0
x(pB) = 7
x(pB) = 8
x(pB) = 9
x(pB) = 10
x(pB) = 11
x(pB) = 12
x(pB) = 13
x(pB) = 14
x(pB) = 15
x(pB) = 16
x(pA) = 17
x(pA) = 18
x(pA) = 19
x(pA) = 0
x(pA) = 1
x(pA) = 2
x(pA) = 3
x(pA) = 4
```

Hình 8: Tương tự khi Process B kết thúc.

Giải thích kết quả

biến mutex đóng vai trò là chìa khóa đóng mở vùng tranh chấp, tránh tình trạng process lấy giá trị của biến trong vùng tranh chấp khi 1 process đang chạy làm dữ liệu trở nên không đồng nhất.

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:

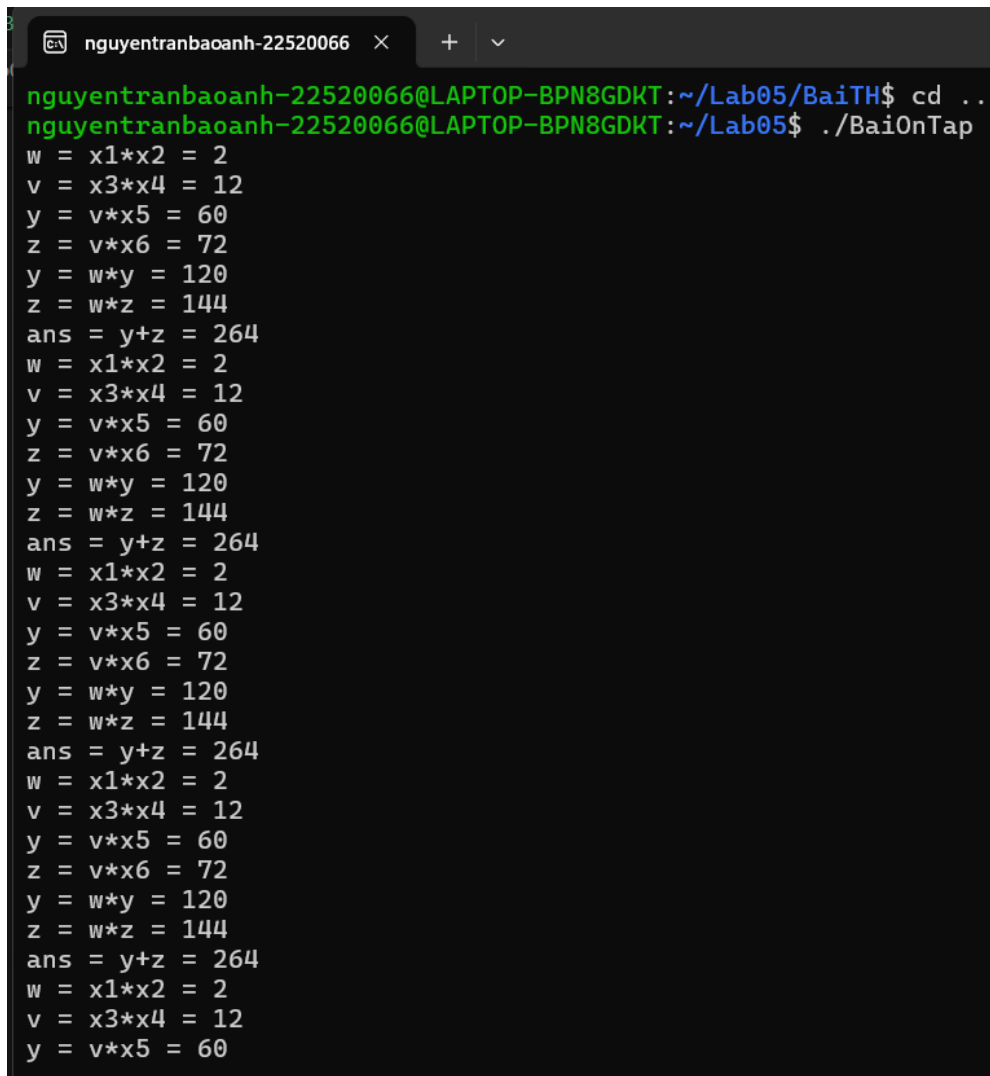
- ✚ (c), (d) chỉ được thực hiện sau khi v được tính
- ✚ (e) chỉ được thực hiện sau khi w và y được tính
- ✚ (g) chỉ được thực hiện sau khi y và z được tính

Trình bày cách làm

```
C Bai2a.c U    C Bai3.c U    C Bai4.c U    C BaiOnTap.c
Lab05 > C BaiOnTap.c > ProcessEF(void *)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <semaphore.h>
4  #include <pthread.h>
5
6  int x1=1, x2=2, x3=3, x4=4, x5=5, x6=6;
7  int w=0, v=0, y=0, z=0, ans=0;
8  sem_t semAB, semCD, semCD_extra, semEF, semEF_extra, semG, busy;
9
10 void* ProcessAB(void* mess)
11 {
12     while(1)
13     {
14         sem_wait(&semG);
15         sem_wait(&busy);
16         w=x1*x2;
17         v=x3*x4;
18         printf("w = x1*x2 = %d\n", w);
19         printf("v = x3*x4 = %d\n", v);
20         sem_post(&semCD_extra);
21         sem_post(&semAB);
22         sem_post(&busy);
23     }
24 }
25
26 void* ProcessCD(void* mess)
27 {
28     while(1)
29     {
30         sem_wait(&semCD_extra);
31         sem_wait(&semAB);
32         sem_wait(&busy);
33         y=v*x5;
34         z=v*x6;
35         printf("y = v*x5 = %d\n", y);
36         printf("z = v*x6 = %d\n", z);
37         sem_post(&semAB);
38         sem_post(&semCD);
39         sem_post(&semEF_extra);
40         sem_post(&busy);
41     }
42 }
```

```
43
44 void* ProcessEF(void* mess)
45 {
46     while(1)
47     {
48         sem_wait(&semEF_extra);
49         sem_wait(&semCD);
50         sem_wait(&semAB);
51         sem_wait(&busy);
52         y=w*y;
53         z=w*z;
54         printf("y = w*y = %d\n", y);
55         printf("z = w*z = %d\n", z);
56         sem_post(&semEF);
57         sem_post(&busy);
58     }
59 }
60
61 void* ProcessG(void* mess)
62 {
63     while(1)
64     {
65         sem_wait(&semEF);
66         sem_wait(&busy);
67         ans = y+z;
68         printf("ans = y+z = %d\n", ans);
69         sem_post(&semG);
70         sem_post(&busy);
71     }
72 }
73
74 int main()
75 {
76     pthread_t pAB, pCD, pEF, pG;
77     sem_init(&semAB, 0, 0);
78     sem_init(&semCD, 0, 0);
79     sem_init(&semCD_extra, 0, 0);
80     sem_init(&semEF, 0, 0);
81     sem_init(&semEF, 0, 0);
82     sem_init(&semG, 0, 1);
83     sem_init(&busy, 0, 1);
84     pthread_create(&pAB, NULL, &ProcessAB, NULL);
85     pthread_create(&pCD, NULL, &ProcessCD, NULL);
86     pthread_create(&pEF, NULL, &ProcessEF, NULL);
87     pthread_create(&pG, NULL, &ProcessG, NULL);
88     while(1){}
89
90     return 0;
91 }
```


Chụp hình minh chứng



```
nguyentranbaoanh-22520066 x + v
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab05/BaiTH$ cd ..
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab05$ ./BaiOnTap
w = x1*x2 = 2
v = x3*x4 = 12
y = v*x5 = 60
z = v*x6 = 72
y = w*y = 120
z = w*z = 144
ans = y+z = 264
w = x1*x2 = 2
v = x3*x4 = 12
y = v*x5 = 60
z = v*x6 = 72
y = w*y = 120
z = w*z = 144
ans = y+z = 264
w = x1*x2 = 2
v = x3*x4 = 12
y = v*x5 = 60
z = v*x6 = 72
y = w*y = 120
z = w*z = 144
ans = y+z = 264
w = x1*x2 = 2
v = x3*x4 = 12
y = v*x5 = 60
z = v*x6 = 72
y = w*y = 120
z = w*z = 144
ans = y+z = 264
w = x1*x2 = 2
v = x3*x4 = 12
y = v*x5 = 60
```

- Process a) và b) được chạy trước.
- Lần lượt các process c) d) e) g) được chạy.

Giải thích kết quả

- Các biến sem_AB, sem_CD, sem_EF và sem_G là các biến semaphore đóng vai trò điều kiện giúp các process khác được chạy. Các biến sem_CD_extra và sem_EF_extra đóng vai trò điều kiện lần lượt cho các process CD và EF được chạy. Biến busy đóng vai trò là mutex, khóa và mở vùng tranh chấp. Khi chương trình được nạp, Giả sử các Process khác AB được chạy trước:

- + Process G sẽ bị khóa bởi biến sem_EF (khởi tạo=0).
- + Process EF sẽ bị khóa bởi biến sem_EF_extra (khởi tạo=0).
- + Process CD sẽ bị khóa bởi biến sem_CD_extra (khởi tạo=0).
- + Process AB sẽ được chạy do biến semG được khởi tạo =1 và khóa vùng tranh chấp.
- + Sau khi AB chạy xong, các điều kiện sẽ được kích hoạt để CD có thể chạy (Các điều kiện này không xung đột với điều kiện ở các process còn lại), đồng thời mở vùng tranh chấp.
- + Sau khi CD chạy xong, tiếp tục tương tự với AB khi cho EF chạy tiếp.
- + EF chạy xong kích hoạt điều kiện cho G được chạy.
- + G sau khi chạy xong, kích hoạt điều kiện để cho A được chạy tiếp, tiếp tục chương trình như 1 vòng lặp.