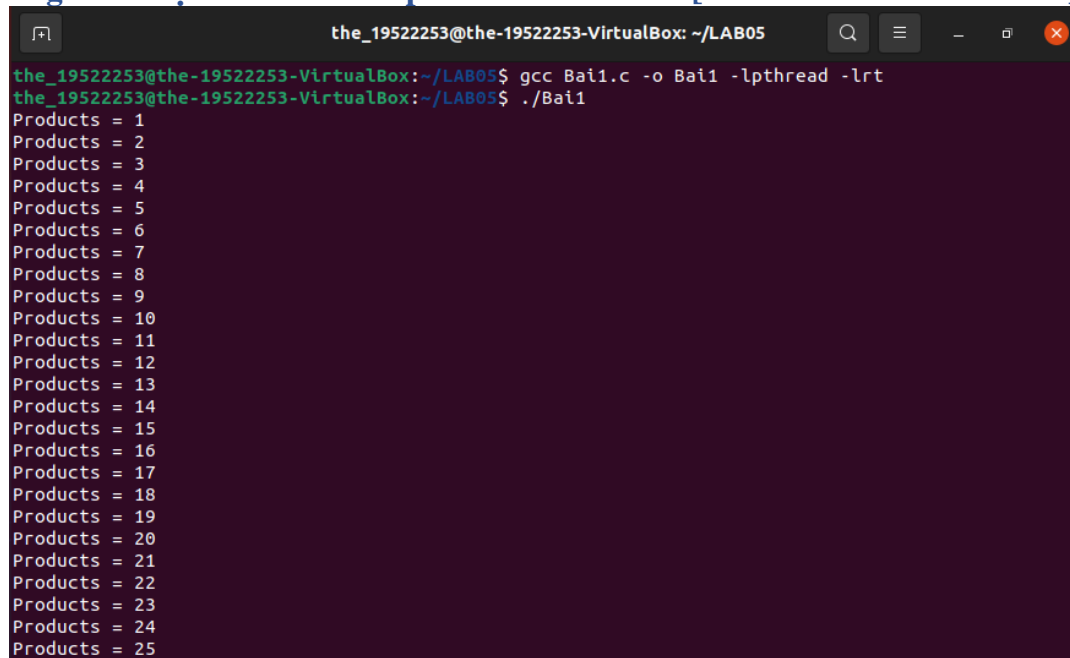


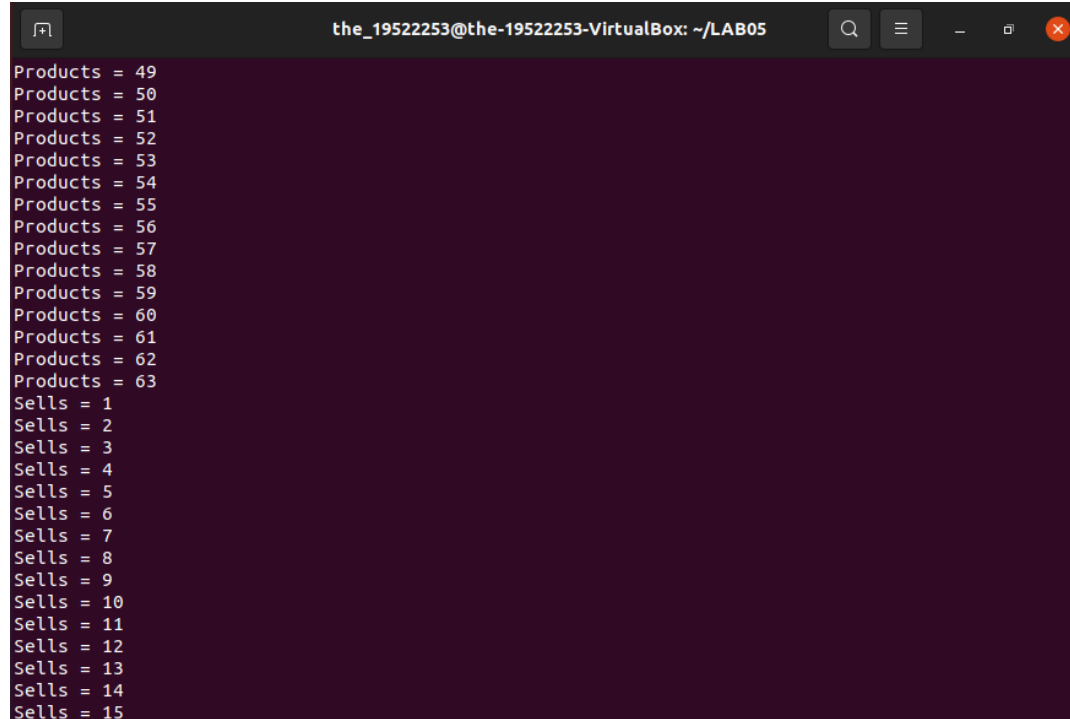
Section 5.4

1. Task name 1: Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau: $\text{sells} \leq \text{products} \leq \text{sells} + [2 \text{ số cuối của MSSV} + 10]$



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
the_19522253@the-19522253-VirtualBox:~/LAB05$ gcc Bai1.c -o Bai1 -lpthread -lrt
the_19522253@the-19522253-VirtualBox:~/LAB05$ ./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
Products = 23
Products = 24
Products = 25
```

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



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
Products = 49
Products = 50
Products = 51
Products = 52
Products = 53
Products = 54
Products = 55
Products = 56
Products = 57
Products = 58
Products = 59
Products = 60
Products = 61
Products = 62
Products = 63
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
Sells = 14
Sells = 15
```

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

```

1  /*#####
2  # University of Information Technology
3  # IT007 Operating System
4  # Pham Duc The, 19522253
5  # File: Bai1.c
6  #####*/
7
8  #include<stdio.h>
9  #include<semaphore.h>
10 #include<pthread.h>
11
12 int sells = 0, products = 0;
13 sem_t sem1, sem2;
14
15 void *ProcessA(void* mess){
16     while(1){
17         sem_wait(&sem1);
18         sells++;
19         printf("Sells = %d\n", sells);
20         sem_post(&sem2);
21     }
22 }
23
24
25 void *ProcessB(void* mess){
26     while(1){
27         sem_wait(&sem2);
28         products++;
29         printf("Products = %d\n", products);
30         sem_post(&sem1);
31     }
32 }
33
34
35 int main(){
36     sem_init(&sem1, 0, 0);
37     sem_init(&sem2, 0, 63);
38     pthread_t pA, pB;
39     pthread_create(&pA, NULL, &ProcessA, NULL);
40     pthread_create(&pB, NULL, &ProcessB, NULL);
41     while(1){}
42     return 0;
43 }
44

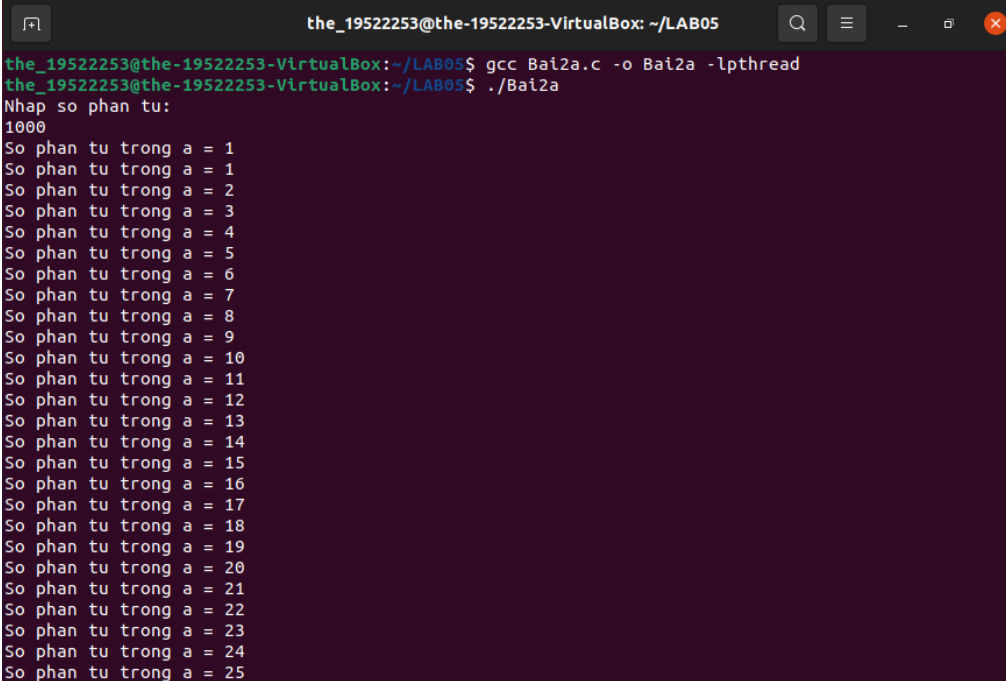
```

Hình 3: Source code bài 1

- **Giải thích:** sem1 đóng vai trò là điều kiện ($sells \leq products$), sem2 đóng vai trò là điều kiện ($products \leq sells + 63$). 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.

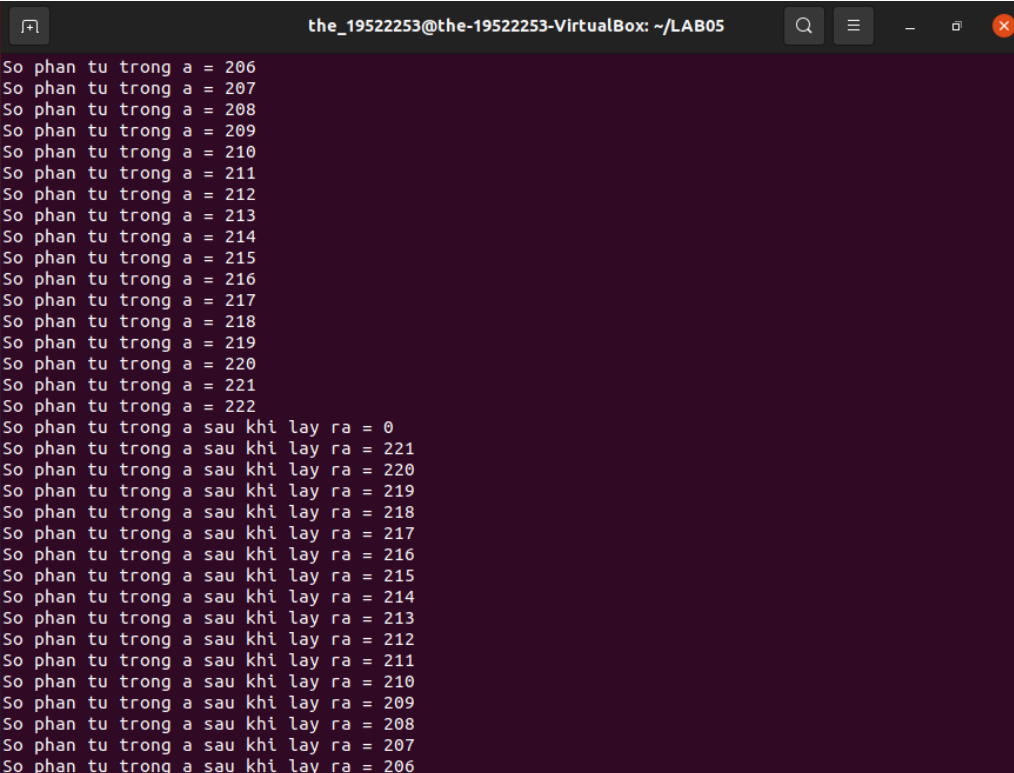
2. Task name 2: Chạy thử và tìm ra lỗi khi chạy chương trình 2 thread song song khi:

a. Chưa được đồng bộ:



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
the_19522253@the-19522253-VirtualBox:~/LAB05$ gcc Bai2a.c -o Bai2a -lpthread
the_19522253@the-19522253-VirtualBox:~/LAB05$ ./Bai2a
Nhap so phan tu:
1000
So phan tu trong a = 1
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 = 11
So phan tu trong a = 12
So phan tu trong a = 13
So phan tu trong a = 14
So phan tu trong a = 15
So phan tu trong a = 16
So phan tu trong a = 17
So phan tu trong a = 18
So phan tu trong a = 19
So phan tu trong a = 20
So phan tu trong a = 21
So phan tu trong a = 22
So phan tu trong a = 23
So phan tu trong a = 24
So phan tu trong a = 25
```

Hình 4: Tiến trình sinh ra phần tử và bỏ vào a được chạy trước.



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
So phan tu trong a = 206
So phan tu trong a = 207
So phan tu trong a = 208
So phan tu trong a = 209
So phan tu trong a = 210
So phan tu trong a = 211
So phan tu trong a = 212
So phan tu trong a = 213
So phan tu trong a = 214
So phan tu trong a = 215
So phan tu trong a = 216
So phan tu trong a = 217
So phan tu trong a = 218
So phan tu trong a = 219
So phan tu trong a = 220
So phan tu trong a = 221
So phan tu trong a = 222
So phan tu trong a sau khi lay ra = 0
So phan tu trong a sau khi lay ra = 221
So phan tu trong a sau khi lay ra = 220
So phan tu trong a sau khi lay ra = 219
So phan tu trong a sau khi lay ra = 218
So phan tu trong a sau khi lay ra = 217
So phan tu trong a sau khi lay ra = 216
So phan tu trong a sau khi lay ra = 215
So phan tu trong a sau khi lay ra = 214
So phan tu trong a sau khi lay ra = 213
So phan tu trong a sau khi lay ra = 212
So phan tu trong a sau khi lay ra = 211
So phan tu trong a sau khi lay ra = 210
So phan tu trong a sau khi lay ra = 209
So phan tu trong a sau khi lay ra = 208
So phan tu trong a sau khi lay ra = 207
So phan tu trong a sau khi lay ra = 206
```

Hình 5: Lỗi xuất hiện khi chưa đồng bộ.

```

1  /*#####
2  # University of Information Technology
3  # IT007 Operating System
4  # Pham Duc The, 19522253
5  # File: Bai2a.c
6  #####*/
7
8  #include<stdio.h>
9  #include<stdlib.h>
10 #include<pthread.h>
11 #include<time.h>
12
13 int* a;
14 int n;
15 int iNum = 0;
16
17 void Arrange(int *a, int x){
18     if(x == iNum){
19         iNum--;
20     }
21     else{
22         for(int i = x; i < iNum-1; i++){
23             a[i] = a[i+1];
24         }
25         iNum--;
26     }
27 }
28
29 void* ProcessA(void* mess){
30     while(1){
31         srand((int)time(0));
32         a[iNum] = rand();
33         iNum++;
34         printf("So phan tu trong a = %d\n", iNum);
35     }
36 }
37
38 void* ProcessB(void* mess){
39     while(1){
40         srand((int)time(0));
41         if(iNum == 0){
42             printf("Nothing in array\n");
43         }
44         else{
45             int r = rand() % iNum;
46             Arrange(a, r);
47             printf("So phan tu trong a sau khi lay ra = %d\n", iNum);
48         }
49     }
50 }
51
52 int main(){
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     return 0;
61 }
62

```

Hình 6: Soucre code bài 2a

b. Đã được đồng bộ:

```
the_19522253@the-19522253-VirtualBox: ~/LAB05
the_19522253@the-19522253-VirtualBox:~/LAB05$ gcc Bai2b.c -o Bai2b -lpthread -lrt
the_19522253@the-19522253-VirtualBox:~/LAB05$ ./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 = 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
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
```

Hình 7: Số phần tử trong *a* được thêm vào và lấy ra luôn có sự đồng nhất về dữ liệu.

```
the_19522253@the-19522253-VirtualBox: ~/LAB05
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
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
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
```

Hình 8: Tương tự khi 2 bên đổi vị trí.

```

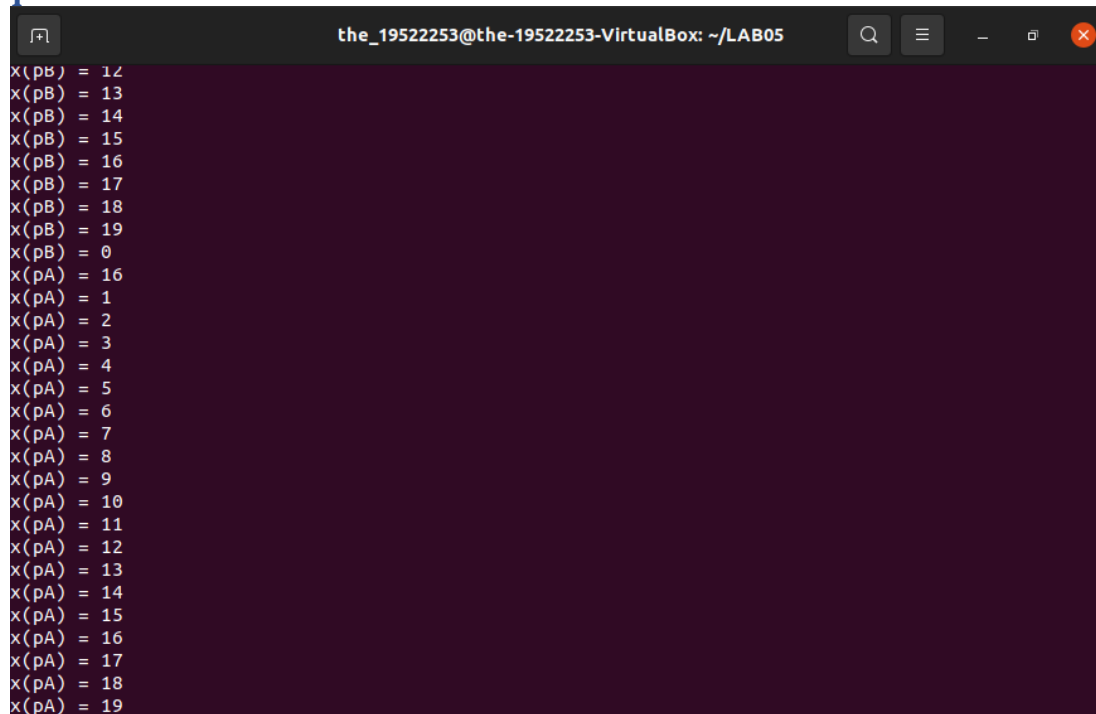
1 /*#####
2 # University of Information Technology
3 # IT007 Operating System
4 # Pham Duc The, 19522253
5 # File: Bai2b.c
6 #####*/
7
8 #include<stdio.h>
9 #include<stdlib.h>
10 #include<pthread.h>
11 #include<time.h>
12 #include<semaphore.h>
13
14 int* a;
15 int n;
16 int iNum = 0;
17 sem_t sem1, sem2, busy;
18
19 void Arrange(int *a, int x){
20     if(x == iNum){
21         iNum--;
22     }
23     else{
24         for(int i = x; i < iNum-1; i++){
25             a[i] = a[i+1];
26         }
27         iNum--;
28     }
29 }
30
31 void* ProcessA(void* mess){
32     while(1){
33         sem_wait(&sem2);
34         sem_wait(&busy);
35         srand((int)time(0));
36         a[iNum] = rand();
37         iNum++;
38         printf("So phan tu trong a = %d\n", iNum);
39         sem_post(&sem1);
40         sem_post(&busy);
41     }
42 }
43
44 void* ProcessB(void* mess){
45     while(1){
46         sem_wait(&sem1);
47         sem_wait(&busy);
48         srand((int)time(0));
49         if(iNum == 0){
50             printf("Nothing in array\n");
51         }
52         else{
53             int r = rand() % iNum;
54             Arrange(a, r);
55             printf("So phan tu trong a sau khi lay ra = %d\n", iNum);
56             sem_post(&sem2);
57             sem_post(&busy);
58         }
59     }
60 }
61
62 int main(){
63     printf("Nhap so phan tu: \n");
64     scanf("%d", &n);
65     a = (int*)malloc(n*sizeof(int));
66     sem_init(&sem1,0,0);
67     sem_init(&sem2,0,n);
68     sem_init(&busy,0,1);
69     pthread_t pA, pB;
70     pthread_create(&pA, NULL, &ProcessA, NULL);
71     pthread_create(&pB, NULL, &ProcessB, NULL);
72     while(1){}
73     return 0;
74 }
75

```

Hình 9: Soucre code bài 2b

- **Giải thích:** 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. Task name 3: Hiện thực mô hình trong hệ điều hành Linux và nhận xét kết quả.



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
x(pB) = 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
x(pA) = 16
x(pA) = 1
x(pA) = 2
x(pA) = 3
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(pA) = 13
x(pA) = 14
x(pA) = 15
x(pA) = 16
x(pA) = 17
x(pA) = 18
x(pA) = 19
```

Hình 10: Kết quả chạy code bài 3

Lỗi xuất hiện khi chưa được đồng bộ. Process A đã lấy giá trị của biến x ở thời điểm Process B đang tính toán x=15, do đó sau khi Process B kết thúc quá trình chạy thì Process A lập tức in ra kết quả đã tính trước đó.

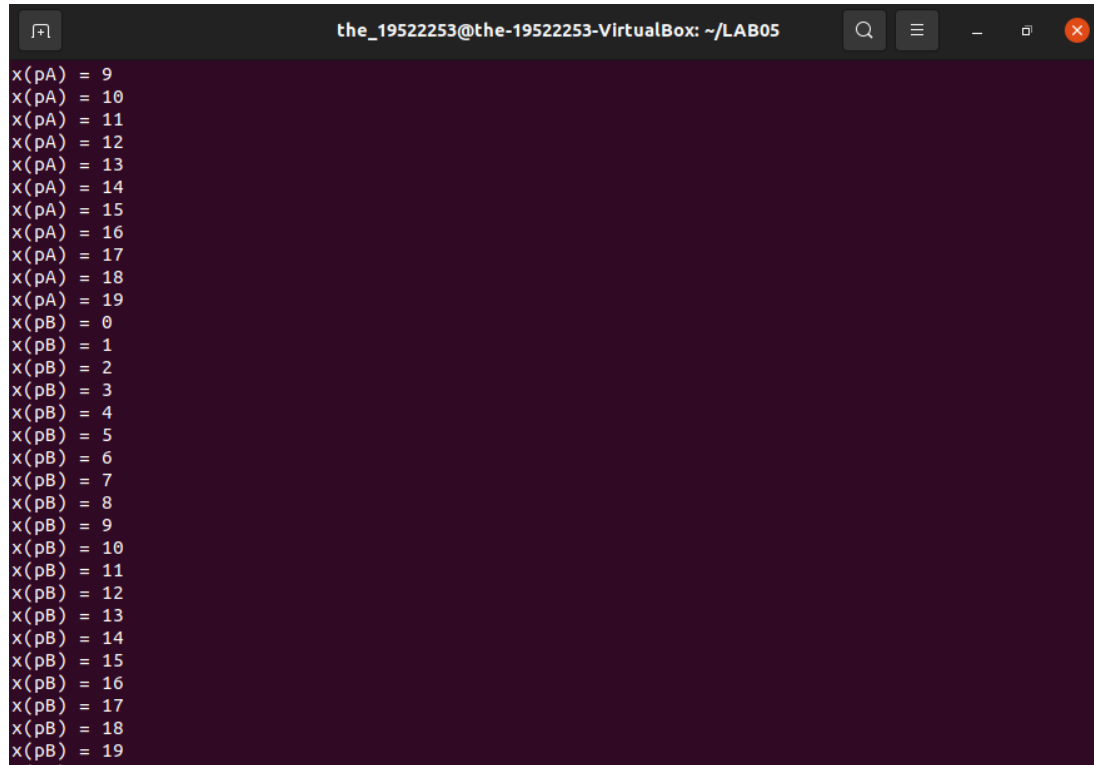
```

1 /*#####
2 # University of Information Technology
3 # IT007 Operating System
4 # Pham Duc The, 19522253
5 # File: Bai3.c
6 #####*/
7
8 #include<stdio.h>
9 #include<semaphore.h>
10 #include<pthread.h>
11
12 int x = 0;
13
14 void* ProcessA(void* mess){
15     while(1){
16         x = x + 1;
17         if(x==20)
18             x = 0;
19         printf("x(pA) = %d\n", x);
20     }
21 }
22
23 void* ProcessB(void* mess){
24     while(1){
25         x = x + 1;
26         if(x==20)
27             x = 0;
28         printf("x(pB) = %d\n", x);
29     }
30 }
31
32 int main(){
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 }
39

```

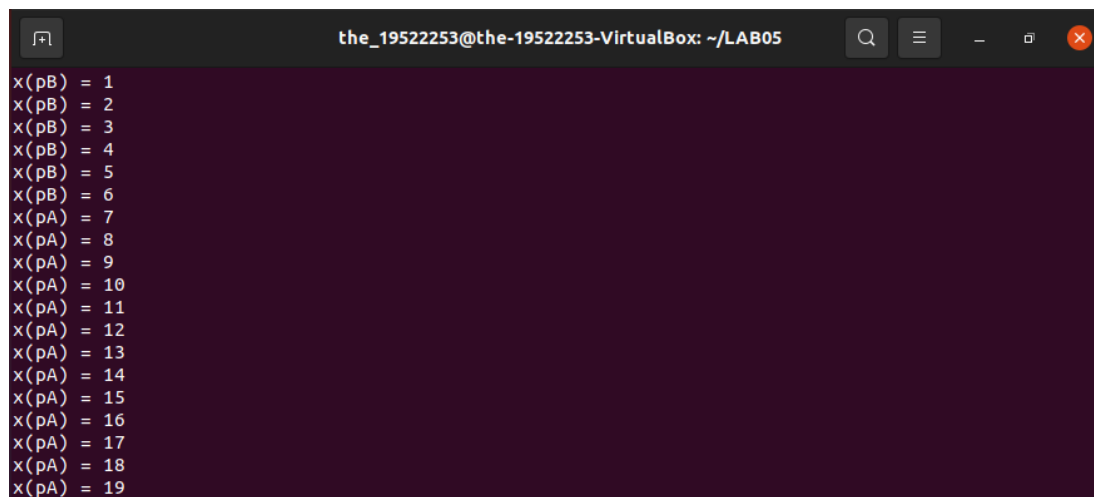
Hình 11: Soucre code bài 3

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



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
x(pA) = 9
x(pA) = 10
x(pA) = 11
x(pA) = 12
x(pA) = 13
x(pA) = 14
x(pA) = 15
x(pA) = 16
x(pA) = 17
x(pA) = 18
x(pA) = 19
x(pB) = 0
x(pB) = 1
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) = 18
x(pB) = 19
```

Hình 12: Process A kết thúc và lập tức Process B tiếp tục tính toán theo kết quả cuối cùng của Process A



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
x(pB) = 1
x(pB) = 2
x(pB) = 3
x(pB) = 4
x(pB) = 5
x(pB) = 6
x(pA) = 7
x(pA) = 8
x(pA) = 9
x(pA) = 10
x(pA) = 11
x(pA) = 12
x(pA) = 13
x(pA) = 14
x(pA) = 15
x(pA) = 16
x(pA) = 17
x(pA) = 18
x(pA) = 19
```

Hình 13: Tương tự khi Process B kết thúc.

```

1 /*#####
2 # University of Information Technology
3 # IT007 Operating System
4 # Pham Duc The, 19522253
5 # File: Bai4.c
6 #####*/
7
8 #include<stdio.h>
9 #include<stdlib.h>
10 #include<pthread.h>
11
12 int x = 0;
13 pthread_mutex_t mutex;
14
15 void* ProcessA(void* mess){
16     while(1){
17         pthread_mutex_lock(&mutex);
18         x = x + 1;
19         if(x==20)
20             x = 0;
21         printf("x(pA) = %d\n", x);
22         pthread_mutex_unlock(&mutex);
23     }
24 }
25
26 void* ProcessB(void* mess){
27     while(1){
28         pthread_mutex_lock(&mutex);
29         x = x + 1;
30         if(x==20)
31             x = 0;
32         printf("x(pB) = %d\n", x);
33         pthread_mutex_unlock(&mutex);
34     }
35 }
36
37 int main(){
38     pthread_t pA, pB;
39     pthread_mutex_init(&mutex, NULL);
40     pthread_create(&pA, NULL, &ProcessA, NULL);
41     pthread_create(&pB, NULL, &ProcessB, NULL);
42     while(1){}
43     return 0;
44 }
45

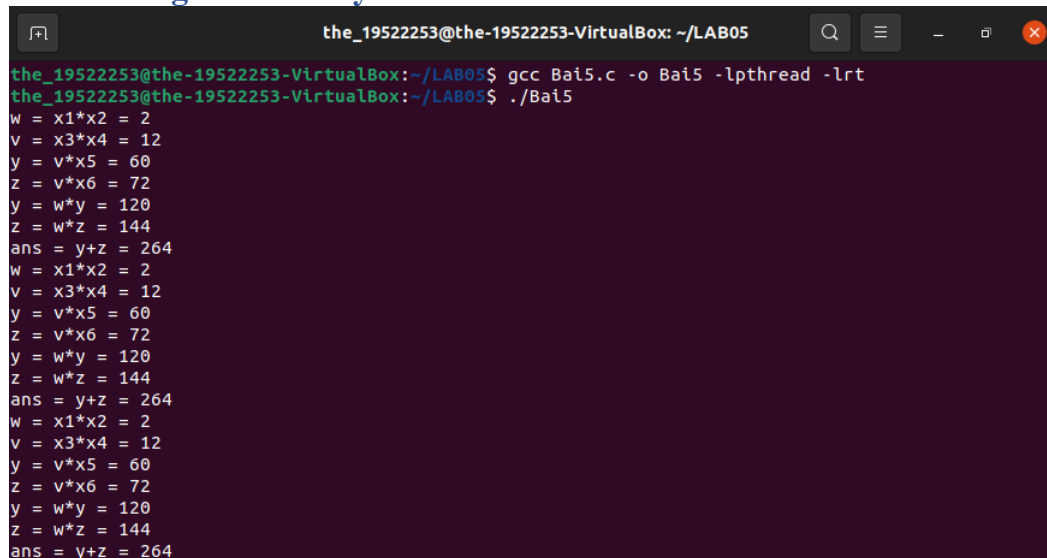
```

Hình 14: Soucre code bài 4

- **Giải thích:** 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.

Section 5.5

1. Task name 1: Lập trình mô phỏng và đồng bộ trên C trong hệ điều hành Linux chương trình theo yêu cầu đề bài:



```
the_19522253@the-19522253-VirtualBox: ~/LAB05
the_19522253@the-19522253-VirtualBox:~/LAB05$ gcc Bai5.c -o Bai5 -lpthread -lrt
the_19522253@the-19522253-VirtualBox:~/LAB05$ ./Bai5
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
```

Hình 15: Kết quả thực thi source code bài 1

Process a) và b) được chạy trước.

Lần lượt các process c) d) e) g) được chạy.

- **Giải thích:** 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.

```

1  /*#####
2  # University of Information Technology
3  # IT007 Operating System
4  # Pham Duc The, 19522253
5  # File: Bai5.c
6  #####*/
7
8  #include<stdio.h>
9  #include<stdlib.h>
10 #include<semaphore.h>
11 #include<pthread.h>
12
13 int x1=1, x2=2, x3=3, x4=4, x5=5, x6=6, w=0, v=0, y=0, z=0, ans=0;
14 sem_t semAB, semCD, semCD_extra, semEF, semEF_extra, semG, busy;
15
16 void* ProcessAB(void* mess){
17     while(1){
18         sem_wait(&semG);
19         sem_wait(&busy);
20         w = x1*x2;
21         v = x3*x4;
22         printf("w = x1*x2 = %d\n", w);
23         printf("v = x3*x4 = %d\n", v);
24         sem_post(&semCD_extra);
25         sem_post(&semAB);
26         sem_post(&busy);
27     }
28 }
29
30 void* ProcessCD(void* mess){
31     while(1){
32         sem_wait(&semCD_extra);
33         sem_wait(&semAB);
34         sem_wait(&busy);
35         y = v*x5;
36         z = v*x6;
37         printf("y = v*x5 = %d\n", y);
38         printf("z = v*x6 = %d\n", z);
39         sem_post(&semAB);
40         sem_post(&semCD);
41         sem_post(&semEF_extra);
42         sem_post(&busy);
43     }
44 }
45
46 void* ProcessEF(void* mess){
47     while(1){
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     while(1){
63         sem_wait(&semEF);
64         sem_wait(&busy);
65         ans = y+z;
66         printf("ans = y+z = %d\n", ans);
67         sem_post(&semG);
68         sem_post(&busy);
69     }
70 }
71
72 int main(){
73     pthread_t pAB, pCD, pEF, pG;
74     sem_init(&semAB, 0, 0);
75     sem_init(&semCD, 0, 0);
76     sem_init(&semCD_extra, 0, 0);
77     sem_init(&semEF, 0, 0);
78     sem_init(&semEF_extra, 0, 0);
79     sem_init(&semG, 0, 1);
80     sem_init(&busy, 0, 1);
81     pthread_create(&pAB, NULL, &ProcessAB, NULL);
82     pthread_create(&pCD, NULL, &ProcessCD, NULL);
83     pthread_create(&pEF, NULL, &ProcessEF, NULL);
84     pthread_create(&pG, NULL, &ProcessG, NULL);
85     while(1){}
86     return 0;
87 }
88

```

Hình 16: Source code bài 1