

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



HỆ ĐIỀU HÀNH – THỰC HÀNH

BÁO CÁO LAB05

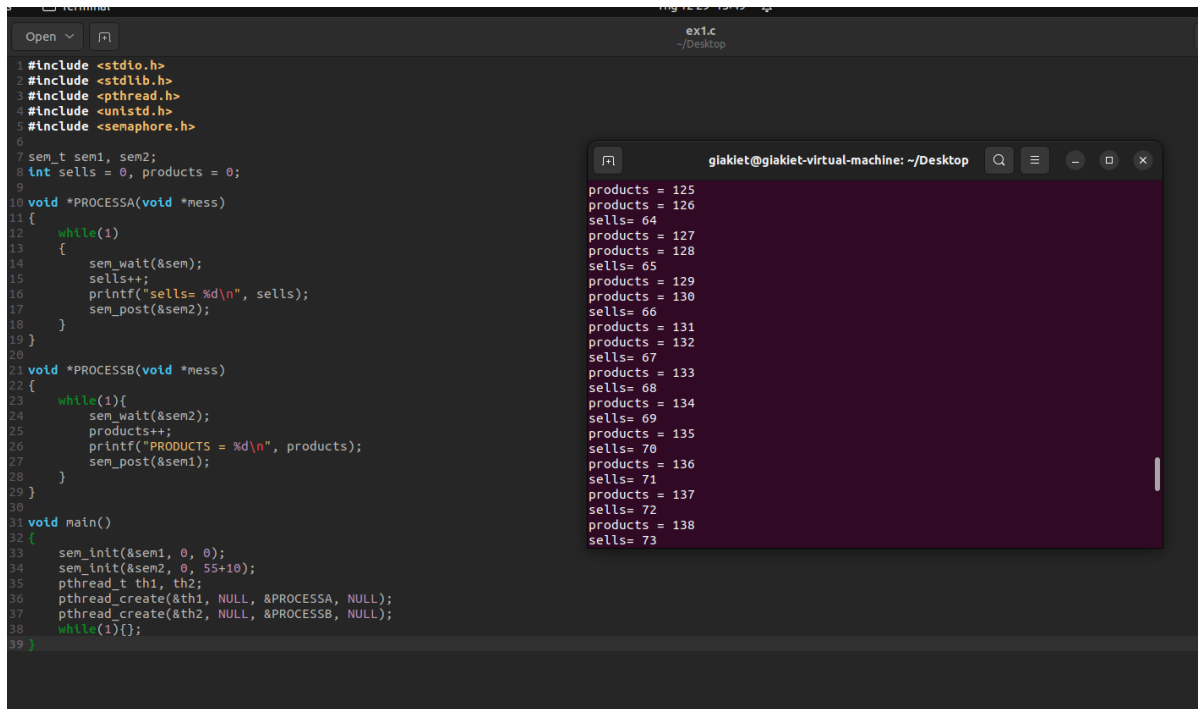
Lớp: IT007.N12.KHCL

Tên: Lê Gia Kiệt

MSSV: 21522255

BÀI LÀM

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



```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>

sem_t sem1, sem2;
int sells = 0, products = 0;

void *PROCESSA(void *mess)
{
    while(1)
    {
        sem_wait(&sem1);
        sells++;
        printf("sells= %d\n", sells);
        sem_post(&sem2);
    }
}

void *PROCESSB(void *mess)
{
    while(1){
        sem_wait(&sem2);
        products++;
        printf("PRODUCTS = %d\n", products);
        sem_post(&sem1);
    }
}

void main()
{
    sem_init(&sem1, 0, 0);
    sem_init(&sem2, 0, 55+10);
    pthread_t th1, th2;
    pthread_create(&th1, NULL, &PROCESSA, NULL);
    pthread_create(&th2, NULL, &PROCESSB, NULL);
    while(1){};
}
```

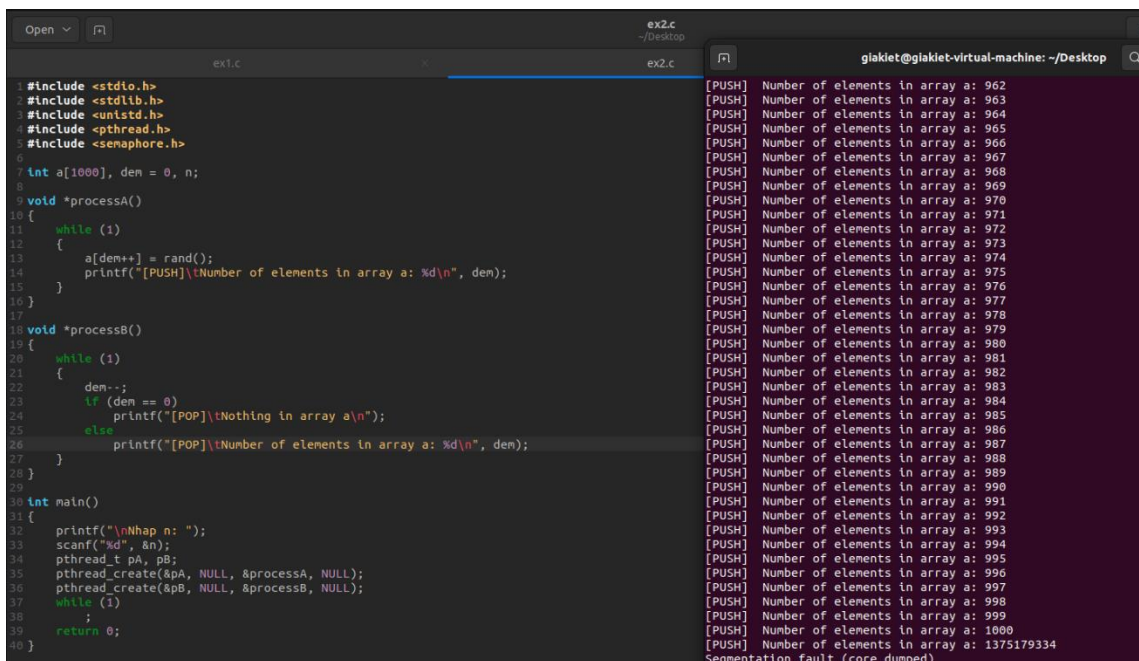
```
products = 125
products = 126
sells= 64
products = 127
products = 128
sells= 65
products = 129
products = 130
sells= 66
products = 131
products = 132
sells= 67
products = 133
sells= 68
products = 134
sells= 69
products = 135
sells= 70
products = 136
sells= 71
products = 137
sells= 72
products = 138
sells= 73
```

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.

BÀI LÀM

- Theo yêu cầu đề bài, tạo ra chương trình như hình phía bên dưới (**hình 1**)
- Vì chưa có semaphore nên chương trình chưa được đồng bộ (**Lỗi logic: cần dùng semaphore để xử lý lỗi này**). B lấy ra kích thước của mảng A khi A chưa được vào chương trình → **xảy ra lỗi**.



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

int a[1000], dem = 0, n;

void *processA()
{
    while (1)
    {
        a[dem++] = rand();
        printf("[PUSH]\tNumber of elements in array a: %d\n", dem);
    }
}

void *processB()
{
    while (1)
    {
        dem--;
        if (dem == 0)
            printf("[POP]\tNothing in array a\n");
        else
            printf("[POP]\tNumber of elements in array a: %d\n", dem);
    }
}

int main()
{
    printf("\nNhap n: ");
    scanf("%d", &n);
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &processA, NULL);
    pthread_create(&pB, NULL, &processB, NULL);
    while (1)
    {
        ;
    }
    return 0;
}
```

```
[PUSH] Number of elements in array a: 962
[PUSH] Number of elements in array a: 963
[PUSH] Number of elements in array a: 964
[PUSH] Number of elements in array a: 965
[PUSH] Number of elements in array a: 966
[PUSH] Number of elements in array a: 967
[PUSH] Number of elements in array a: 968
[PUSH] Number of elements in array a: 969
[PUSH] Number of elements in array a: 970
[PUSH] Number of elements in array a: 971
[PUSH] Number of elements in array a: 972
[PUSH] Number of elements in array a: 973
[PUSH] Number of elements in array a: 974
[PUSH] Number of elements in array a: 975
[PUSH] Number of elements in array a: 976
[PUSH] Number of elements in array a: 977
[PUSH] Number of elements in array a: 978
[PUSH] Number of elements in array a: 979
[PUSH] Number of elements in array a: 980
[PUSH] Number of elements in array a: 981
[PUSH] Number of elements in array a: 982
[PUSH] Number of elements in array a: 983
[PUSH] Number of elements in array a: 984
[PUSH] Number of elements in array a: 985
[PUSH] Number of elements in array a: 986
[PUSH] Number of elements in array a: 987
[PUSH] Number of elements in array a: 988
[PUSH] Number of elements in array a: 989
[PUSH] Number of elements in array a: 990
[PUSH] Number of elements in array a: 991
[PUSH] Number of elements in array a: 992
[PUSH] Number of elements in array a: 993
[PUSH] Number of elements in array a: 994
[PUSH] Number of elements in array a: 995
[PUSH] Number of elements in array a: 996
[PUSH] Number of elements in array a: 998
[PUSH] Number of elements in array a: 999
[PUSH] Number of elements in array a: 1000
[PUSH] Number of elements in array a: 1375179334
Segmentation fault (core dumped)
```

Hình 1

→ Chương trình sau khi được fix bằng cách sử dụng semaphore(hình 2: code & hình 3: kết quả)



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

sem_t sem1, sem2; // Khởi tạo 2 semaphore sem1 và sem2, sem1 quản lý số lần push phần tử vào mảng a, sem2 quản lý số lần pop phần tử ra khỏi mảng a.

pthread_mutex_t mutex; // Khởi tạo mutex để tránh trường hợp 2 process cùng vào vùng tranh chấp.

int a[1000], dem = 0, n; // Mảng a chứa các phần tử được push vào, dem là số lượng phần tử trong mảng a, n là số lượng phần tử tối đa có thể push vào mảng a.

void *processA() // processA sẽ push phần tử vào mảng
{
    while (1) // vòng lặp vô hạn
    {
        sem_wait(&sem1) // sem_wait(&sem1) sẽ giảm giá trị của sem1 đi 1, nếu giá trị của sem1 = 0 thì processA sẽ bị block cho đến khi giá trị của sem1 > 0.

        pthread_mutex_lock(&mutex); // Khóa mutex để tránh trường hợp 2 process cùng vào vùng tranh chấp.

        a[dem++] = rand(); // Push phần tử ngẫu nhiên vào mảng a.

        printf("[PUSH]\tNumber of elements in array a: %d\n", dem); // In ra số lượng phần tử trong mảng a.

        pthread_mutex_unlock(&mutex); // Mở khóa mutex.

        sem_post(&sem2); // sem_post(&sem2) sẽ tăng giá trị của sem2 lên 1.
    }
}

void *processB() // processB sẽ pop phần tử ra khỏi mảng a
{
    while (1) // vòng lặp vô hạn
    {
        sem_wait(&sem2); // sem_wait(&sem2) sẽ giảm giá trị của sem2 đi 1, nếu giá trị của sem2 = 0 thì processB sẽ bị block cho đến khi giá trị của sem2 > 0.

        pthread_mutex_lock(&mutex); // Khóa mutex để tránh trường hợp 2 process cùng vào vùng tranh chấp.

        dem--; // Pop phần tử ra khỏi mảng a.

        if (dem == 0) // Kiểm tra xem mảng a có phần tử nào không?
            printf("[POP]\tNothing in array a\n"); // Nếu không có phần tử nào thì in ra "Nothing in array a".
        else // Ngược lại.
            printf("[POP]\tNumber of elements in array a: %d\n", dem); // In ra số lượng phần tử trong mảng a.

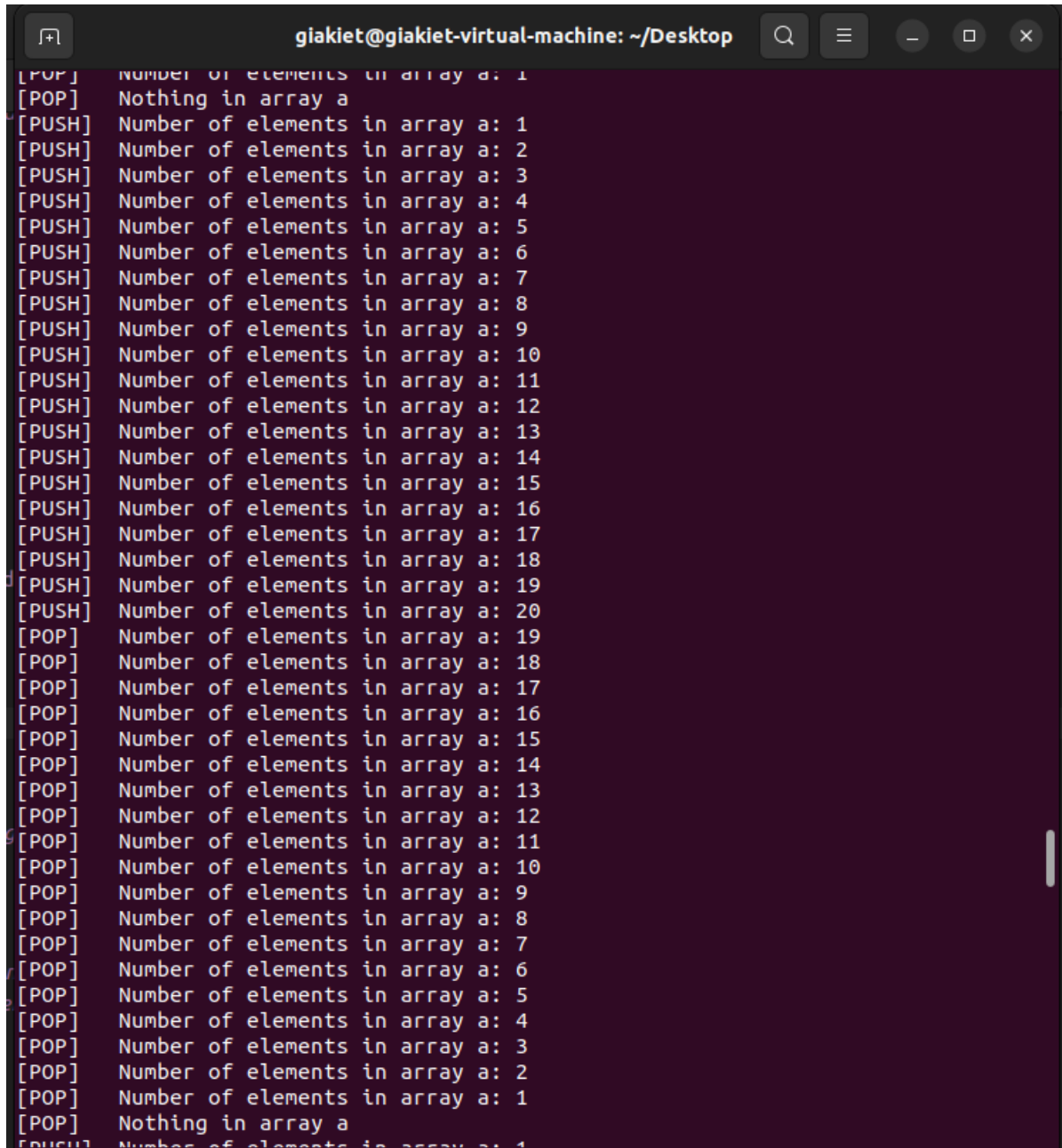
        pthread_mutex_unlock(&mutex); // Mở khóa mutex.

        sem_post(&sem1); // sem_post(&sem1) sẽ tăng giá trị của sem1 lên 1.
    }
}

int main()
{
    printf("\nNhap n: ");
    scanf("%d", &n); // Nhập số lượng phần tử tối đa có thể push vào mảng a.
    sem_init(&sem1, 0, n); // Khởi tạo sem1 với giá trị n (số lượng phần tử tối đa có thể push vào mảng a).
    sem_init(&sem2, 0, 0); // Khởi tạo sem2 với giá trị 0.
    pthread_t pA, pB; // Khởi tạo mutex.
    pthread_create(&pA, NULL, &processA, NULL); // Khởi tạo 2 thread pA và pB.
    pthread_create(&pB, NULL, &processB, NULL); // Tạo thread pA.
    while (1) // Tạo thread pB.
    {
        ;
    }
    return 0;
}
```

Hình 2

```
giakiet@giakiet-virtual-machine:~/Desktop$ gcc ex2.c -o ex2 -lpthread -lrt
giakiet@giakiet-virtual-machine:~/Desktop$ ./ex2
```



```
giakiet@giakiet-virtual-machine: ~/Desktop
[POP] Number of elements in array a: 1
[POP] Nothing in array a
[PUSH] Number of elements in array a: 1
[PUSH] Number of elements in array a: 2
[PUSH] Number of elements in array a: 3
[PUSH] Number of elements in array a: 4
[PUSH] Number of elements in array a: 5
[PUSH] Number of elements in array a: 6
[PUSH] Number of elements in array a: 7
[PUSH] Number of elements in array a: 8
[PUSH] Number of elements in array a: 9
[PUSH] Number of elements in array a: 10
[PUSH] Number of elements in array a: 11
[PUSH] Number of elements in array a: 12
[PUSH] Number of elements in array a: 13
[PUSH] Number of elements in array a: 14
[PUSH] Number of elements in array a: 15
[PUSH] Number of elements in array a: 16
[PUSH] Number of elements in array a: 17
[PUSH] Number of elements in array a: 18
[PUSH] Number of elements in array a: 19
[PUSH] Number of elements in array a: 20
[POP] Number of elements in array a: 19
[POP] Number of elements in array a: 18
[POP] Number of elements in array a: 17
[POP] Number of elements in array a: 16
[POP] Number of elements in array a: 15
[POP] Number of elements in array a: 14
[POP] Number of elements in array a: 13
[POP] Number of elements in array a: 12
[POP] Number of elements in array a: 11
[POP] Number of elements in array a: 10
[POP] Number of elements in array a: 9
[POP] Number of elements in array a: 8
[POP] Number of elements in array a: 7
[POP] Number of elements in array a: 6
[POP] Number of elements in array a: 5
[POP] Number of elements in array a: 4
[POP] Number of elements in array a: 3
[POP] Number of elements in array a: 2
[POP] Number of elements in array a: 1
[POP] Nothing in array a
[PUSH] Number of elements in array a: 1
```

Hình 3: Kết quả sau khi thực thi code

Nhận xét: Khi compile code ta dùng tag *-lpthread* và *-lrt* để có thể kích hoạt **semaphore** và **mutex** trong source code, sau khi thực thi ta thấy số lượng phần tử không bao giờ vượt quá n phần tử cụ thể trong lần thực thi này là $n = 20$, khi đã pop hết phần tử ra mảng a thì in ra “*Nothing in array a*”

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

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

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

- Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

int x = 0;
void *thread_PA(void *thread)
{
    while (1)
    {
        x++;
        if (x == 20)
            x = 0;
        printf("PA | x = %d\n", x);
    }
}

void *thread_PB(void *thread)
{
    while (1)
    {
        x++;
        if (x == 20)
            x = 0;
        printf("PB | x = %d\n", x);
    }
}

int main()
{
    pthread_t threadA, threadB;
    pthread_create(&threadA, NULL, &thread_PA, NULL);
    pthread_create(&threadB, NULL, &thread_PB, NULL);
    while (1)
    {
        ;
    }
    return 0;
}
```

- Kết quả

```
giakiet@giakiet-virtual-ma
PB | x = 9
PB | x = 10
PB | x = 11
PB | x = 12
PB | x = 13
PB | x = 14
PB | x = 15
PB | x = 16
PB | x = 17
PB | x = 18
PB | x = 19
PB | x = 0
PB | x = 1
PB | x = 2
PB | x = 3
PB | x = 4
PB | x = 5
PA | x = 10
PA | x = 6
PA | x = 7
PA | x = 8
PA | x = 9
PA | x = 10
PA | x = 11
PA | x = 12
PA | x = 13
PA | x = 14
PA | x = 15
PA | x = 16
PA | x = 17
PA | x = 18
PA | x = 19
PA | x = 0
PA | x = 1
PA | x = 2
PA | x = 3
PA | x = 4
PA | x = 5
PA | x = 6
PA | x = 7
PA | x = 8
PA | x = 9
```

- Nhận xét:** 2 process chạy đồng thời nhưng không đồng bộ do khi 1 process vào vùng tranh chấp thì process còn lại không bị block mà vẫn được vào xảy ra vi phạm loại trừ tương hỗ nên 2 process không đồng bộ.

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

- Cách thực hiện

- Kết quả

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

pthread_mutex_t mutex;          // khởi tạo mutex để tránh trường hợp 2 process cùng vào vùng tranh chấp
int x = 0;                      // khởi tạo biến x = 0

void *thread_PA(void *thread)   // thread_PA() là thread tăng số lượng x
{
    while (1)                  // vòng lặp vô hạn
    {
        pthread_mutex_lock(&mutex); // khóa mutex để tránh trường hợp 2 process cùng vào vùng tranh chấp
        x++;                      // tăng số lượng x
        if (x == 20)              // kiểm tra x có bằng 20 hay không
            x = 0;                // nếu bằng 20 thì gán x = 0
        printf("PA | x = %d\n", x); // in ra số lượng x
        pthread_mutex_unlock(&mutex); // mở khóa mutex để cho phép các process khác vào vùng tranh chấp
    }
}

void *thread_PB(void *thread)   // thread_PB() là thread tăng số lượng x
{
    while (1)                  // vòng lặp vô hạn
    {
        pthread_mutex_lock(&mutex); // khóa mutex để tránh trường hợp 2 process cùng vào vùng tranh chấp
        x++;                      // tăng số lượng x
        if (x == 20)              // kiểm tra x có bằng 20 hay không
            x = 0;                // nếu bằng 20 thì gán x = 0
        printf("PB | x = %d\n", x); // in ra số lượng x
        pthread_mutex_unlock(&mutex); // mở khóa mutex để cho phép các process khác vào vùng tranh chấp
    }
}

int main()
{
    pthread_mutex_init(&mutex, NULL); // khởi tạo mutex
    pthread_t threadA, threadB;        // khởi tạo 2 thread
    pthread_create(&threadA, NULL, &thread_PA, NULL); // tạo threadA
    pthread_create(&threadB, NULL, &thread_PB, NULL); // tạo threadB
    while (1)
    ;
    return 0;
}
```

```
PA | x = 7
PA | x = 8
PA | x = 9
PA | x = 10
PA | x = 11
PA | x = 12
PA | x = 13
PA | x = 14
PA | x = 15
PA | x = 16
PA | x = 17
PA | x = 18
PA | x = 19
PA | x = 0
PA | x = 1
PA | x = 2
PA | x = 3
PA | x = 4
PA | x = 5
PA | x = 6
PA | x = 7
PB | x = 8
PB | x = 9
PB | x = 10
PB | x = 11
PB | x = 12
PB | x = 13
PB | x = 14
PB | x = 15
PB | x = 16
PB | x = 17
PB | x = 18
PB | x = 19
PB | x = 0
PB | x = 1
PB | x = 2
PB | x = 3
```

```
giakiet@giakiet-virtual-machine:~/Desktop$ gcc ex4.c -o ex4 -lpthread -lrt
giakiet@giakiet-virtual-machine:~/Desktop$ ./ex4
```

- Nhận xét:** Khi compile code ta dùng tag *-lpthread* và *-lrt* để có thể kích hoạt mutex trong source code, sau khi thực thi ta thấy chỉ processA hoặc processB được vào vùng tranh chấp do sử dụng mutex và có thể thấy được sự đồng bộ giữa 2 tiến trình

5.5 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

BÀI LÀM

- Thực thi code và kết quả thu được đúng yêu cầu đề bài

```
giakiet@giakiet-virtual-machine:~/Desktop$ gcc ex5.c -o ex5 -lpthread -lrt
giakiet@giakiet-virtual-machine:~/Desktop$ ./ex5
Enter x1, x2, x3, x4, x5, x6:
1 2 3 4 5 6
PB | v = 3 * 4 = 12
PD | z = 12 * 6 = 72
PC | y = 12 * 5 = 60
PA | w = 1 * 2 = 2
PE | y = 2 * 60 = 120
PF | z = 2 * 72 = 144
PG | ans = 120 + 144 = 264
giakiet@giakiet-virtual-machine:~/Desktop$
```

- Source code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

int x1, x2, x3, x4, x5, x6, w, v, y, z, ans;
/*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)*/
sem_t semc, semd, seme1, seme2, semf1, semf2, semg1, semg2; // khởi tạo semaphore để quản lý các process

void *processA(void *thread) // processA() là process tính w
{
    w = x1 * x2; // tính w
    printf("PA | w = %d * %d = %d\n", x1, x2, w); // in ra w
    sem_post(&seme1); // mở khóa seme1 để cho phép processE() vào vùng tranh chấp
    sem_post(&semf1); // mở khóa semf1 để cho phép processF() vào vùng tranh chấp
}

void *processB(void *thread) // processB() là process tính v
{
    v = x3 * x4; // tính v
    printf("PB | v = %d * %d = %d\n", x3, x4, v); // in ra v
    sem_post(&semc); // mở khóa semc để cho phép processC() vào vùng tranh chấp
    sem_post(&semd); // mở khóa semd để cho phép processD() vào vùng tranh chấp
}

void *processC(void *thread) // processC() là process tính y
{
    sem_wait(&semc); // đợi cho đến khi semc được mở khóa
    y = v * x5; // tính y
    printf("PC | y = %d * %d = %d\n", v, x5, y); // in ra y
    sem_post(&seme2); // mở khóa seme2 để cho phép processE() vào vùng tranh chấp
}

void *processD(void *thread) // processD() là process tính z
{
    sem_wait(&semd); // đợi cho đến khi semd được mở khóa
    z = v * x6; // tính z
    printf("PD | z = %d * %d = %d\n", v, x6, z); // in ra z
    sem_post(&semf2); // mở khóa semf2 để cho phép processF() vào vùng tranh chấp
}

void *processE(void *thread) // processE() là process tính y
{
    sem_wait(&seme1); // đợi cho đến khi seme1 được mở khóa
    sem_wait(&seme2); // đợi cho đến khi seme2 được mở khóa
    y = w * y; // tính y
    printf("PE | y = %d * %1.f = %d\n", w, (float)y / w, y); // in ra y
    sem_post(&semg1); // mở khóa semg1 để cho phép processG() vào vùng tranh chấp
}

void *processF(void *thread) // processF() là process tính z
{
    sem_wait(&semf1); // đợi cho đến khi semf1 được mở khóa
    sem_wait(&semf2); // đợi cho đến khi semf2 được mở khóa
    z = w * z; // tính z
    printf("PF | z = %d * %1.f = %d\n", w, (float)z / w, z); // in ra z
    sem_post(&semg2); // mở khóa semg2 để cho phép processG() vào vùng tranh chấp
}

void *processG(void *thread) // processG() là process tính ans
{
    sem_wait(&semg1); // đợi cho đến khi semg1 được mở khóa
    sem_wait(&semg2); // đợi cho đến khi semg2 được mở khóa
    ans = y + z; // tính ans
    printf("PG | ans = %d + %d = %d\n", y, z, ans); // in ra ans
}

int main()
{
    printf("Enter x1, x2, x3, x4, x5, x6:\n");
    scanf("%d %d %d %d %d", &x1, &x2, &x3, &x4, &x5, &x6); // nhập các giá trị x1, x2, x3, x4, x5, x6
    pthread_t threadA, threadB, threadC, threadD, threadE, threadF, threadG; // khai báo các thread
    sem_init(&semc, 0, 0); // khởi tạo các semaphore
    sem_init(&semd, 0, 0);
    sem_init(&seme1, 0, 0);
    sem_init(&seme2, 0, 0);
    sem_init(&semf1, 0, 0);
    sem_init(&semf2, 0, 0);
    sem_init(&semg1, 0, 0);
    sem_init(&semg2, 0, 0);
    pthread_create(&threadA, NULL, &processA, NULL); // tạo các thread
    pthread_create(&threadB, NULL, &processB, NULL);
    pthread_create(&threadC, NULL, &processC, NULL);
    pthread_create(&threadD, NULL, &processD, NULL);
    pthread_create(&threadE, NULL, &processE, NULL);
    pthread_create(&threadF, NULL, &processF, NULL);
    pthread_create(&threadG, NULL, &processG, NULL);
    pthread_join(threadA, NULL); // đợi các thread kết thúc
    pthread_join(threadB, NULL);
    pthread_join(threadC, NULL);
    pthread_join(threadD, NULL);
    pthread_join(threadE, NULL);
    pthread_join(threadF, NULL);
    pthread_join(threadG, NULL);
    return 0;
}
```