BÁO CÁO LAB5

Họ và tên: Lâm Thiên Phát
MSSV: 22521068

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]

Sử dụng hai semaphore để đảm bảo điều kiện `sells <= products <= sells + 1068`.

- Semaphore 1 (sem1) khởi tạo bằng giá trị `products - sells`, đại diện cho khoảng giá trị mà `sells` có thể tăng.

- Semaphore 2 (sem2) khởi tạo bằng giá trị `sells + 1068 - products`, đại diện cho khoảng giá trị mà `products` có thể tăng.

Tiểu trình `sells`

1. Chờ sem1 có giá trị dương (`products > sells`) bằng `sem_wait(&sem1)`.

2. Tăng `sells` lên 1.

3. Tăng giá trị của sem2 lên 1 (`sem_post(&sem2)`).

Tiểu trình `products`

1. Chờ sem2 có giá trị dương (`products < sells + 1068`) bằng `sem_wait(&sem2)`.

2. Tăng `products` lên 1.

3. Tăng giá trị của sem1 lên 1 (`sem_post(&sem1)`).

```c
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
int sells = 0;
int products = 0;
sem_t sem1, sem2;
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);

    }
}
int main(){
    sem_init(&sem1, 0 , products -sells);
    sem_init(&sem2, 0 , sells+1068-products);
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &ProcessA, NULL);
    pthread_create(&pB, NULL, &ProcessB, NULL);
    while (1)
    {
        /* code */
    }
    return 0;

}
```

```
sells = 688901
sells = 688902
sells = 688903
products = 688904
products = 688905
sells = 688904
sells = 688905
products = 688906
products = 688907
products = 688908
products = 688909
products = 688910
products = 688911
products = 688912
products = 688913
products = 688914
products = 688915
products = 688916
products = 688917
products = 688918
products = 688919
products = 688920
products = 688921
products = 688922
products = 688923
products = 688924
products = 688925
products = 688926
products = 688927
products = 688928
products = 688929
products = 688930
products = 688931
products = 688932
```

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.

Khi chưa đồng bộ

Cách làm

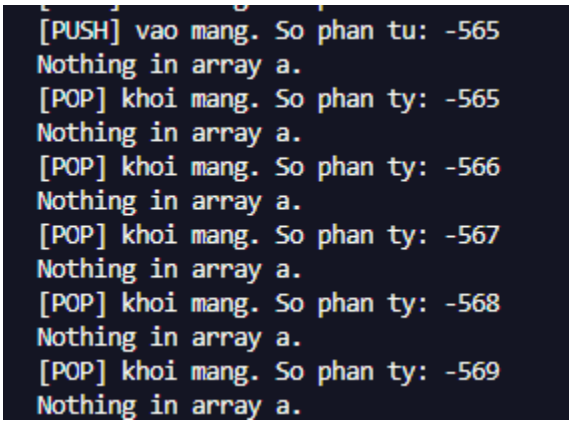
- Cấp phát động cho con trỏ a một vùng nhớ để lưu trữ n số nguyên (n do người dung nhập vào)
- Hai biến n và count là 2 biến toàn cục, n là số phần tử của mảng a do người dung nhập vào. count được gán giá trị ban đầu bằng 0, dùng để kiểm tra số lượng phần tử trong mảng.
- Tạo ra hai thread chạy song song là p1,p2

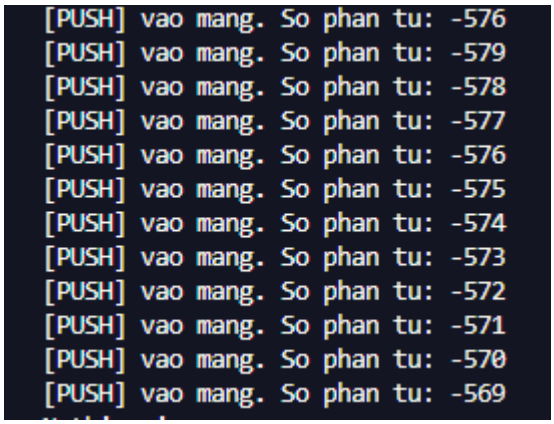
• P1 sẽ tạo một số nguyên ngẫu nhiên từ 0 đến 99 sau đó đưa vào mảng a, tăng biến count lên 1 đơn vị và in ra màn hình

• P2 sẽ lấy ra một phần tử trong a (ở đây em chọn lấy ra phần tử cuối cùng trong mảng), lưu lại giá trị của phần tử bị xóa, giảm biến count đi 1 đơn vị và xuất ra màn hình. Nếu lúc này, count nhỏ hơn hoặc bằng 0 (trong mảng không còn phần tử nào) thì xuất ra dòng chữ “Nothing in array a”.

```c
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <pthread.h>
```

```c
#include <semaphore.h>

#include <unistd.h>

int *a;

int n, count = 0;

void *P1(void *mess) {
  while (1) {
    srand(time(NULL));
    int randNumber = rand() % 100;
    a[count] = randNumber;
    count++;
    printf("\n[PUSH] vao mang. So phan tu: %d", count);
    //sleep(1);
  }
}

void *P2(void *mess) {
  while (1) {
    int deleted = a[count - 1];
    a[count - 1] = 0;
    count--;
    printf("\n[POP] khoi mang. So phan ty: %d", count);

    if (count <= 0) {
      printf("\nNothing in array a.");
    }
    //sleep(1);
  }
}

int main() {
  pthread_t p1, p2;
  printf("Nhap n: ");
  scanf("%d", &n);
  a = (int *)malloc(n * sizeof(int));
  pthread_create(&p1, NULL, &P1, NULL);
  pthread_create(&p2, NULL, &P2, NULL);
  while (1){}
    ;

  return 0;
}
```





Giải thích

Khi chạy đoạn chương trình trên sẽ có các vấn đề xảy ra:

- Ta thấy ở tiểu trình P2, do chúng ta thiếu rang buộc điều kiện nên xảy ra vấn đề, dù trong a không còn phần tử nào nữa nhưng tiểu trình vẫn thực hiện việc giảm count đi 1 đơn vị, dẫn đến việc count thành số âm
- Bên cạnh đó ở tiểu trình P1 thì lại xảy ra vấn đề về số lượng phần tử. Ban đầu khi nhập n ta chỉ nhập 5, tuy nhiên khi chương trình chạy đến một thời điểm nào đó, số lượng phần tử trong mảng a lại vượt quá n. Vấn đề này xảy ra do chúng ta đã thiếu ràng buộc điều kiện về số lượng phần tử của mảng.
- Tiếp theo là lỗi về vùng tranh chấp. Do cả 2 tiểu trình đều thực hiện đoạn code trong vùng tranh chấp của mình dẫn đến xảy ra lỗi, khiến cho giá trị của biến count tăng giảm. lộn xộn mà không theo yêu cầu tăng hoặc giảm 1 đơn vị.

Khi đã đồng bộ

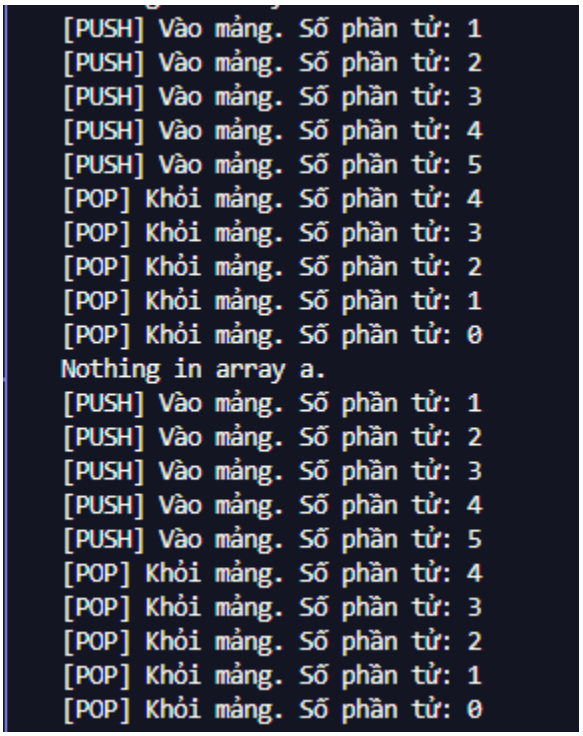
Cách làm

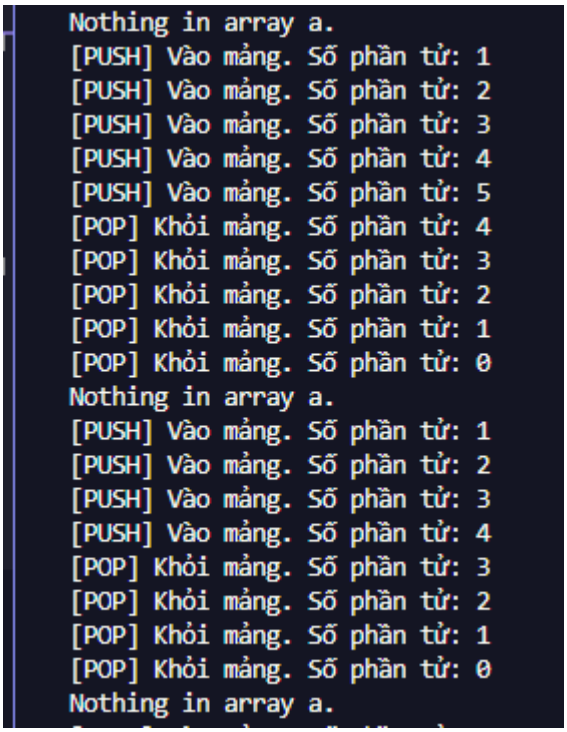
Để đồng bộ bài toán trên, em sẽ dung cả semaphore và khóa mutex. Dùng 2 semaphore để giải quyết lỗi biến count bị âm và số phần tử lớn hơn n, dung mutex để đảm bảo mutual exclusion sao cho tại mỗi thời điểm thì chỉ có 1 tiểu trình đang ở trong vùng tranh chấp.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```c
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

int *a;
int n, count = 0;

sem_t sem1, sem2;
pthread_mutex_t mutex;

void* P1(void* mess) {
    while (1) {
        sem_wait(&sem2);
        pthread_mutex_lock(&mutex);
        srand(time(NULL));
        int randNumber = rand() % 100;
        a[count] = randNumber;
        count++;
        printf("\n [PUSH] Vào mảng. Số phần tử: %d", count);
        // sleep(1);
        sem_post(&sem1);
        pthread_mutex_unlock(&mutex);
    }
}

void* P2(void* mess) {
    while (1) {
        sem_wait(&sem1);
        pthread_mutex_lock(&mutex);
        int deleted = a[count - 1];
        a[count - 1] = 0;
        count--;
        printf("\n [POP] Khỏi mảng. Số phần tử: %d", count);
        if (count <= 0) {
            printf("\n Nothing in array a.");
        }
        // sleep(1);
        sem_post(&sem2);
        pthread_mutex_unlock(&mutex);
    }
}

int main() {
    pthread_t p1, p2;
    printf("Nhập n: ");
```

```
    scanf("%d", &n);
    a = (int*)malloc(n * sizeof(int));
    sem_init(&sem1, 0, 0);
    sem_init(&sem2, 0, n);
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&p1, NULL, P1, NULL);
    pthread_create(&p2, NULL, P2, NULL);
    while (1);
    return 0;
}
```





Giải thích

- Ban đầu nhập n=10

- Quan sát kết quả ta thấy không có 1 thời điểm nào số phần tử trong mảng a vượt quá 10

- Khi số phần tử trong mảng còn 0 thì chương trình kh thể thực hiện xóa đi 1 phần tử nữa mà phải chờ cho tới khi mảng có thêm phần tử thì mới được xóa.

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

int x = 0;	
PROCESS A	PROCESS B

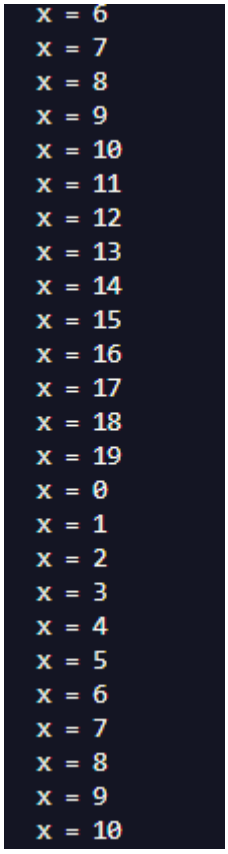
processA()
{
while(1){
x = x + 1;
if (x == 20)
x = 0;
print(x);
}
}
	processB()
	{
	while(1){
	x = x + 1;
	if (x == 20)
	x = 0;
	print(x);
	}
	}

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

Cách làm

```c
#include<stdio.h>
#include<pthread.h>
int x = 0;
void* processA(void * mess){
    while (1){
        x = x+1;
        if(x==20)
        x = 0;
        printf("x = %d\n", x);
    }
}
void * processB(void * mess){
    while (1)
    {
        x= x+1;
        if(x==20)
        x = 0;
        printf("x = %d\n", x);
    }

}
int main(){
```

```
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &processA, NULL);
    pthread_create(&pA, NULL, &processB, NULL);
    while (1)
    {
        /* code */
    }
    return 0;

}
```



Giải thích kết quả

- Mục tiêu ban đầu là tạo ra 2 tiểu trình chạy luân phiên nhau sau cho x được in ra màn hình có giá trị từ 0 đến 19 liên tục

- Tuy nhiên khi quan sát kết quả chạy code ta thấy giá trị của x bị lỗi ở nhiều chỗ. Như ở hình trên, x đang có giá trị 13, tăng lên 14 và theo như ý đồ của người lập trình thì x phải tiếp tục tăng lên 15. Tuy nhiên kết quả x xuất ra màn hình lại bằng 0 rồi sau đó mới đến 15.

- Lỗi này xảy ra do hai tiểu trình cùng tiến vào vùng tranh chấp, dẫn đến xung đột và xảy

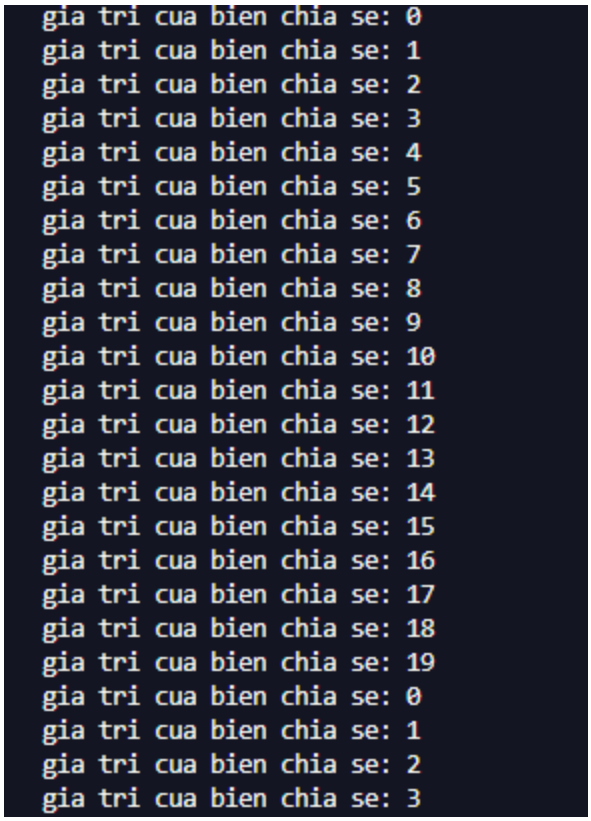
ra lỗi.

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

Cách làm

Sử dụng khóa mutex để đảm bảo tại mỗi thời điểm chỉ có 1 tiến trình tiến vào vùng tranh chấp. Như vậy, vấn đề đã đề cập trong câu 3 sẽ được giải quyết

```c
#include<stdio.h>
#include<pthread.h>
 int shared_variable  = 0;
 pthread_mutex_t mutex;
 void * thread_function(void *arg){
    while(1){
        pthread_mutex_lock(&mutex);
        shared_variable++;
        if(shared_variable == 20){
            shared_variable = 0;
        }
        printf("gia tri cua bien chia se: %d\n", shared_variable);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
 }
 int main ()
 {
    pthread_mutex_init(&mutex, NULL);
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, thread_function, NULL);
    pthread_create(&thread2, NULL, thread_function, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    return 0;
 }
```



Giải thích kết quả

- Quan sát kết quả, ta thấy giá trị của x được in lặp đi lặp lại từ 0-19 mà không có chỗ nào bị lỗi

- Như vậy, sau khi sử dụng khóa mutex, ta đã đảm bảo đc mutual exclusion và giải quyết được vấn đề xung đột khi hai tiểu tình cùng tiến vào vùng tranh chấp