TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

MÔN HỆ ĐIỀU HÀNH

BÁO CÁO THỰC HÀNH LAB 5

Lớp IT007.P11

23520552 – PHAN CẢNH ĐĂNG HUÂN

NỘI DUNG BÁO CÁO

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]

Code:

bai1.c > ProcessA(void *)

```c
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
int sells = 0, products = 0;
sem_t sem_sells, sem_products;
// MSSV: 23520552 -> 4 số cuối = 552
const int MAX_DIFF = 552;
void *ProcessA(void* mess) {
    while(1) {
        sem_wait(&sem_sells); // Chờ để đảm bảo điều kiện
        if (sells < products) { // Chỉ tăng sells nếu sells < products
            sells++;
            printf("Sells = %d\n", sells);
        }
        sem_post(&sem_products); // Mở semaphore để ProcessB hoạt động
    }
}
void *ProcessB(void* mess) {
    while(1) {
        sem_wait(&sem_products); // Chờ để đảm bảo điều kiện
        if (products < sells + MAX_DIFF) { // Chỉ tăng products nếu products <= sells + 552
            products++;
            printf("Products = %d\n", products);
        }
        sem_post(&sem_sells); // Mở semaphore để ProcessA hoạt động
    }
}
int main() {
    // Khởi tạo semaphore: ban đầu sem_sells = 1, sem_products = 0
    sem_init(&sem_sells, 0, 1);
    sem_init(&sem_products, 0, 0);
    pthread_t pA, pB;
    // Tạo hai tiến trình ProcessA và ProcessB
    pthread_create(&pA, NULL, &ProcessA, NULL);
    pthread_create(&pB, NULL, &ProcessB, NULL);
    // Vòng lặp vô hạn để các tiến trình chạy
    while(1) {}
    return 0;
}
```

Output:

```
Products = 10381
Sells = 10381
Products = 10382
Sells = 10382
Products = 10383
Sells = 10383
Products = 10384
Sells = 10384
Products = 10385
Sells = 10385
Products = 10386
Sells = 10386
Products = 10387
Sells = 10387
Products = 10388
Sells = 10388
Products = 10389
Sells = 10389
Products = 10390
Sells = 10390
Products = 10391
Sells = 10391
```

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:

a. 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.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
int *a;
int n;
int iNum = 0;
void Arrange(int *a, int x)
{
    if (x == iNum)
    {
        iNum--;
    }
    else
    {
        for (int i = int i iNum - 1; i++)
        {
            a[i] = a[i + 1];
        }
        iNum--;
    }
}
void *ProcessA(void *mess)
{
    while (1)
    {
        srand((int)time(0));
        a[iNum] = rand();
        iNum++;
        printf("So phan tu trong a = %d\n", iNum);
    }
}
void *ProcessB(void *mess)
{
    while (1)
    {
        srand((int)time(0));
        if (iNum == 0)
        {
            printf("Nothing in array\n");
        }
        else
        {
            int r = rand() % iNum;
            Arrange(a, r);
            printf("So phan tu trong a sau khi lay ra = %d\n", iNum);
        }
    }
}
```

```c
int main()
{
    printf("Nhap so phan tu: \n");
    scanf("%d", &n);
    a = (int *)malloc(n * sizeof(int));
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &ProcessA, NULL);
    pthread_create(&pB, NULL, &ProcessB, NULL);
    while (1)
    {
    }
    return 0;
}
```

Output:

```
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 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 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 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
```

b. 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".

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <semaphore.h>
int *a;
int n;
int iNum = 0;
sem_t sem1, sem2, busy;
void Arrange(int *a, int x)
{
    if (x == iNum)
    {
        iNum--;
    }
    else
    {
        for (int i = x; i < iNum - 1; i++)
        {
            a[i] = a[i + 1];
        }
        iNum--;
    }
}
void *ProcessA(void *mess)
{
    while (1)
    {
        sem_wait(&sem2);
        sem_wait(&busy);
        srand((int)time(0));
        a[iNum] = rand();
        iNum++;
        printf("So phan tu trong a = %d\n", iNum);
        sem_post(&busy);
        sem_post(&sem1);
    }
}
void *ProcessB(void *mess)
{
    while (1)
```

```c
    {
        sem_wait(&sem1);
        sem_wait(&busy);
        srand((int)time(0));
        if (iNum == 0)
        {
            printf("Nothing in array\n");
        }
        else
        {
            int r = rand() % iNum;
            Arrange(a, r);
            printf("So phan tu trong a sau khi lay ra = %d\n", iNum);
        }
        sem_post(&sem2);
        sem_post(&busy);
    }
}
int main()
{
    printf("Nhap so phan tu: \n");
    scanf("%d", &n);
    a = (int *)malloc(n * sizeof(int));
    sem_init(&sem1, 0, 0);
    sem_init(&sem2, 0, n);
    sem_init(&busy, 0, 1);
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &ProcessA, NULL);
    pthread_create(&pB, NULL, &ProcessB, NULL);
    while (1)
    {
    }
    return 0;
}
```

Output:

```
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 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 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 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
```

Bài 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.

Code:

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
int x = 0;
void *ProcessA(void *mess)
{
    while (1)
    {
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("x (pA) = %d\n", x);
    }
}
void *ProcessB(void *mess)
{
    while (1)
    {
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("x (pB) = %d\n", x);
    }
}
int main()
{
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &ProcessA, NULL);
    pthread_create(&pB, NULL, &ProcessB, NULL);
    while (1)
    {
    }
    return 0;
}
```

Output:

```
x (pA) = 19
x (pA) = 0
x (pA) = 1
x (pB) = 17
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) = 12
```

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

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
int x = 0;
pthread_mutex_t mutex;
void *processA()
{
    while (1)
    {
        pthread_mutex_lock(&mutex);
        x = x + 1;
        if (x == 20)
        {
            x = 0;
        }
        printf("[processA] x = %d\n", x);
        pthread_mutex_unlock(&mutex);
    }
}
void *processB()
{
    while (1)
    {
        pthread_mutex_lock(&mutex);
        x = x + 1;
        if (x == 20)
        {
            x = 0;
        }
        printf("[processB] x = %d\n", x);
        pthread_mutex_unlock(&mutex);
    }
}
void main()
{
    pthread_mutex_init(&mutex, NULL);
    pthread_t pr1, pr2;
    pthread_create(&pr1, NULL, &processA, NULL);
    pthread_create(&pr2, NULL, &processB, NULL);
    while (1)
    {
    };
}
```

Output:

```
[processA] x = 7
[processA] x = 8
[processA] x = 9
[processA] x = 10
[processA] x = 11
[processA] x = 12
[processA] x = 13
[processA] x = 14
[processA] x = 15
[processA] x = 16
[processA] x = 17
[processA] x = 18
[processA] x = 19
[processA] x = 0
```