

Name: Hà Danh Tuấn

ID: 20522109

Class: IT007.M13

OPERATING SYSTEM LAB 5'S REPORT

SUMMARY

Task		Status	Page
5.4	Bài 1	Hoàn thành	2
	Bài 2	Hoàn thành	4
	Bài 3	Hoàn thành	9
	Bài 4	Hoàn thành	10
5.5	Bài 1	Hoàn thành	13

Self-scores: 8

5.4

Bài 1: Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau: $\text{sells} \leq \text{products} \leq \text{sells} + 19$

Có 2 điều kiện cần đồng bộ cho bài tập này đó là $\text{sells} \leq \text{products}$ và $\text{products} \leq \text{sells} + 19$

Cần sử dụng 2 semaphore, mỗi semaphore xử lý 1 điều kiện.

Giá trị của semaphore đầu tiên là 0, thể hiện sự chênh lệch của sells và products , nếu giá trị của nó là 0 thì processA sẽ bị block cho đến khi processB chạy được ít nhất 1 lần.

Giá trị của semaphore thứ 2 là 19, thể hiện sự chênh lệch của products và $\text{sells} + 19$, nếu giá trị của nó là 0 tức là $\text{products} = \text{sells} + 19$ thì processB sẽ bị block cho đến khi processA chạy được ít nhất 1 lần.

Code:

```

# include <stdio.h>
# include <semaphore.h>
# include <pthread.h>

int sells = 0, products = 0;
sem_t sem1, sem2;
//sem1=0, sem2=19
void *processA()
{
    while(1)
    {
        sem_wait(&sem1);
        sells++;
        printf("SELLS = %d\n", sells);
        sem_post(&sem2);
    }
}

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

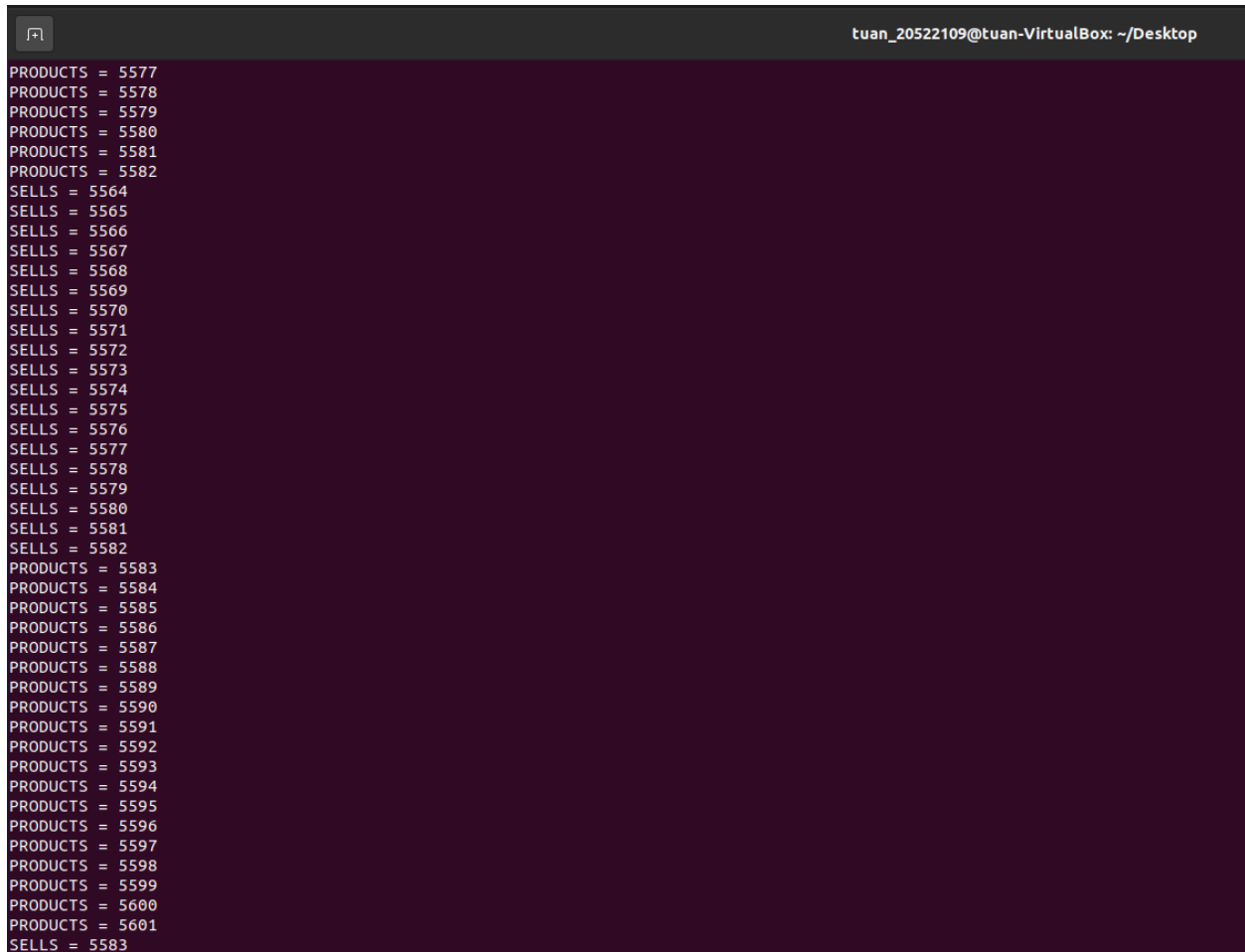
int main()
{
    sem_init(&sem1,0,0);
    sem_init(&sem2,0,19);

    pthread_t A,B;
    pthread_create(&A, NULL, &processA, NULL);
    pthread_create(&B, NULL, &processB, NULL);

    while(1);
}

"bai1.c" 79L, 656C
```

Chạy chương trình:



```
tuan_20522109@tuan-VirtualBox: ~/Desktop
PRODUCTS = 5577
SELLS = 5564
PRODUCTS = 5578
SELLS = 5565
PRODUCTS = 5579
SELLS = 5566
PRODUCTS = 5580
SELLS = 5567
PRODUCTS = 5581
SELLS = 5568
PRODUCTS = 5582
SELLS = 5569
SELLS = 5570
SELLS = 5571
SELLS = 5572
SELLS = 5573
SELLS = 5574
SELLS = 5575
SELLS = 5576
SELLS = 5577
SELLS = 5578
SELLS = 5579
SELLS = 5580
SELLS = 5581
SELLS = 5582
PRODUCTS = 5583
PRODUCTS = 5584
PRODUCTS = 5585
PRODUCTS = 5586
PRODUCTS = 5587
PRODUCTS = 5588
PRODUCTS = 5589
PRODUCTS = 5590
PRODUCTS = 5591
PRODUCTS = 5592
PRODUCTS = 5593
PRODUCTS = 5594
PRODUCTS = 5595
PRODUCTS = 5596
PRODUCTS = 5597
PRODUCTS = 5598
PRODUCTS = 5599
PRODUCTS = 5600
PRODUCTS = 5601
SELLS = 5583
```

Nhận thấy giá trị của sells luôn lớn hơn products và sell luôn bé hơn products + 19

Bài 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”.

Dùng 2 semaphore và 1 mutex để giải quyết bài toán này.

Semaphore sem1 có giá trị khởi tạo là 0 với mục đích kiểm tra xem mảng đã rỗng chưa, nếu giá trị sem1 bằng 0 tức là lúc đó mảng đã rỗng, không thể lấy phần tử ra, thread pop sẽ bị khóa cho đến khi thread push thực hiện ít nhất một lần và thực hiện sem_post(&sem1) để đưa pop ra khỏi trạng thái block.

Semaphore sem2 có giá trị khởi tạo là độ dài tối đa của mảng, nó sẽ kiểm tra xem mảng có đầy hay không trước khi thêm dữ liệu vào.

Mutex giúp các thread truy cập được vào vùng tranh chấp, khi đang thêm hay đang xóa dữ liệu thì thread còn lại không được vào vùng tranh chấp.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
int i = 0;
int a[100000];
pthread_mutex_t mutex;
sem_t sem1, sem2;
void *push()
{
    while(1)
    {
        sem_wait(&sem2);
        pthread_mutex_lock(&mutex);
        a[i++] = rand() % 10;
        printf("\n[PUSH] size: %d", i);
        pthread_mutex_unlock(&mutex);
        sem_post(&sem1);
    }
}
void *pop()
{
    while(1)
    {
        sem_wait(&sem1);
        pthread_mutex_lock(&mutex);
        a[i--] = -1;
        if(i == 0 )
            printf("\nnothing in array a");
        else
            printf("\n[POP] size: %d", i);
        pthread_mutex_unlock(&mutex);
        sem_post(&sem2);
    }
}
int main()
{
    pthread_t A,B;
    int n;
    printf("nhap n: ");
    scanf("%d", &n);
    pthread_mutex_init(&mutex, NULL);
    sem_init(&sem1, 0, 0);
    sem_init(&sem2, 0, n);

    pthread_create(&A, NULL, &push, NULL);
```

Chạy chương trình:

```
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ gcc bai2.c -o bai2 -lpthread -lrt
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ ./bai2
nhap n: 10

[PUSH] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[PUSH] size: 5
[PUSH] size: 6
[PUSH] size: 7
[PUSH] size: 8
[PUSH] size: 9
[PUSH] size: 10
[POP] size: 9
[POP] size: 8
[POP] size: 7
[POP] size: 6
[POP] size: 5
[POP] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
nothing in array a
[PUSH] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[PUSH] size: 5
[PUSH] size: 6
[PUSH] size: 7
[PUSH] size: 8
[PUSH] size: 9
[PUSH] size: 10
[POP] size: 9
[POP] size: 8
```

```

[PUSH] size: 5
[PUSH] size: 6
[PUSH] size: 7
[PUSH] size: 8
[PUSH] size: 9
[PUSH] size: 10
[POP] size: 9
[POP] size: 8
[POP] size: 7
[POP] size: 6
[POP] size: 5
[POP] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
nothing in array a
[PUSH] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[PUSH] size: 5
[PUSH] size: 6
[PUSH] size: 7
[PUSH] size: 8
[PUSH] size: 9
[PUSH] size: 10
[POP] size: 9
[POP] size: 8
[POP] size: 7
[POP] size: 6
[POP] size: 5
[POP] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
nothing in array a
[PUSH] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[PUSH] size: 5
[PUSH] size: 6
[PUSH] size: 7
[PUSH] size: 8
[PUSH] size: 9
[POP] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[PUSH] size: 5
[PUSH] size: 6
[POP] size: 5
[POP] size: 4
[POP] size: 3
[POP] size: 2
[POP] size: 1
[PUSH] size: 2
[PUSH] size: 3
[PUSH] size: 4
[PUSH] size: 5
[PUSH] size: 6

```

Có thể thấy thread push không thêm phần tử vào mảng khi mảng đã đầy và thread pop cũng không lấy phần tử khỏi mảng khi mảng rỗng. Khi không còn phần tử nào trong mảng thì xuất ra màn hình “nothing in array a”.

Bài 3: Hiện thực mô hình trên C trong hệ điều hành Linux và nhận xét kết quả

Code:

```
tuan_20522109@tuan-VirtualBc
#include<stdio.h>
#include<pthread.h>

int x = 0;

void *processA()
{
    while(1)
    {
        x = x + 1;

        if (x == 20)
            x = 0;

        printf("PROCESS A: %d\n", x);
    }
}

void *processB()
{
    while(1)
    {
        x = x + 1;

        if (x == 20)
            x = 0;

        printf("PROCESS B: %d\n", x);
    }
}

int main()
{
    pthread_t A,B;
    pthread_create(&A, NULL, &processA, NULL);
    pthread_create(&B, NULL, &processB, NULL);

    while(1);
}
```

Chạy chương trình:

```

PROCESS B: 8
PROCESS B: 9
PROCESS B: 10
PROCESS B: 11
PROCESS B: 12
PROCESS B: 13
PROCESS B: 14
PROCESS B: 15
PROCESS B: 16
PROCESS B: 17
PROCESS B: 18
PROCESS B: 19
PROCESS B: 0
PROCESS B: 1
PROCESS B: 2
PROCESS B: 3
PROCESS B: 4
PROCESS B: 5
PROCESS B: 6
PROCESS B: 7
PROCESS B: 8
PROCESS B: 9
PROCESS B: 10
PROCESS A: 13
PROCESS A: 11
PROCESS A: 12
PROCESS A: 13
PROCESS A: 14
PROCESS A: 15
PROCESS A: 16
PROCESS A: 17
PROCESS A: 18
PROCESS A: 19
PROCESS A: 0
PROCESS A: 1
PROCESS A: 2

```

Có thể thấy được kết quả in ra không đúng theo thứ tự, lý do là bởi do thread hết time slice khi tới câu lệnh printf.

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

Dùng mutex để đồng bộ, tiến trình nào chạy trước sẽ khóa mutex, thực hiện công việc trong vùng tranh chấp cụ thể là tăng giá trị của x và in ra màn hình, thread còn lại khi gọi hàm lock() thì sẽ bị khóa cho đến khi thread này mở mutex ra. Như vậy sẽ không còn bị bất hợp lý trong kết quả nữa.

```

tuan_20522109@tuan-Virtual

#include<stdio.h>
#include<pthread.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("PROCESS A: %d\n", x);
        pthread_mutex_unlock(&mutex);
    }
}

void *processB()
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("PROCESS B: %d\n", x);
        pthread_mutex_unlock(&mutex);
    }
}

int main()
{
    pthread_t A,B;
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&A, NULL, &processA, NULL);
    pthread_create(&B, NULL, &processB, NULL);

    while(1);
}
```

Chạy chương trình:

```
PROCESS A: 0
PROCESS A: 1
PROCESS A: 2
PROCESS A: 3
PROCESS A: 4
PROCESS A: 5
PROCESS A: 6
PROCESS A: 7
PROCESS A: 8
PROCESS A: 9
PROCESS A: 10
PROCESS A: 11
PROCESS A: 12
PROCESS A: 13
PROCESS A: 14
PROCESS A: 15
PROCESS A: 16
PROCESS A: 17
PROCESS A: 18
PROCESS A: 19
PROCESS A: 0
PROCESS A: 1
PROCESS A: 2
PROCESS A: 3
PROCESS A: 4
PROCESS A: 5
PROCESS A: 6
PROCESS A: 7
PROCESS A: 8
PROCESS A: 9
PROCESS A: 10
PROCESS A: 11
PROCESS A: 12
PROCESS A: 13
PROCESS A: 14
PROCESS A: 15
PROCESS A: 16
PROCESS A: 17
PROCESS A: 18
PROCESS A: 19
PROCESS A: 0
PROCESS A: 1
PROCESS A: 2
PROCESS A: 3
```

```
PROCESS B: 14
PROCESS B: 15
PROCESS B: 16
PROCESS B: 17
PROCESS B: 18
PROCESS B: 19
PROCESS B: 0
PROCESS B: 1
PROCESS B: 2
PROCESS B: 3
PROCESS B: 4
PROCESS B: 5
PROCESS B: 6
PROCESS B: 7
PROCESS B: 8
PROCESS B: 9
PROCESS B: 10
PROCESS B: 11
PROCESS B: 12
PROCESS B: 13
PROCESS B: 14
PROCESS B: 15
PROCESS B: 16
PROCESS B: 17
PROCESS B: 18
PROCESS A: 19
PROCESS A: 0
PROCESS A: 1
PROCESS A: 2
PROCESS A: 3
PROCESS A: 4
PROCESS A: 5
PROCESS A: 6
PROCESS A: 7
PROCESS A: 8
PROCESS A: 9
PROCESS A: 10
PROCESS A: 11
PROCESS A: 12
PROCESS A: 13
PROCESS A: 14
PROCESS A: 15
```

5.5

Bài 5:

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

Để có thể giải quyết bài toán cần có 8 semaphore gồm `semAE`, `semAF`, `semBC`, `semBD`, `semCE`, `semDF`, `semEG`, `semFG`. Giá trị khởi đầu của tất cả semaphore đều bằng 0.

semAE với mục đích thread (a) chạy trước rồi thread (e) mới chạy được, tương tự với các semaphore còn lại.

Code:

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

int x1, x2, x3, x4, x5, x6;
int w, v, y, z, ans;
sem_t semAE, semAF, semBC, semBD, semCE, semDF, semEG, semFG;

void *processA()
{
    w = x1 * x2;
    printf("w = %d\n", w);
    sem_post(&semAE);
    sem_post(&semAF);
}

void *processB()
{
    v = x3 * x4;
    printf("v = %d\n", v);
    sem_post(&semBC);
    sem_post(&semBD);
}

void *processC()
{
    sem_wait(&semBC);
    y = v * x5;
    printf("y = %d\n", y);
    sem_post(&semCE);
}

void *processD()
{
    sem_wait(&semBD);
    z = v * x6;
    printf("z = %d\n", z);
    sem_post(&semDF);
}

void *processE()
{
    sem_wait(&semAE);
    sem_wait(&semCE);
    y = w * y;
    printf("y = %d\n", y);
    sem_post(&semEG);
}
```

```

void *processE()
{
    sem_wait(&semAE);
    sem_wait(&semCE);
    y = w * y;
    printf("y = %d\n", y);
    sem_post(&semEG);
}

void *processF()
{
    sem_wait(&semAF);
    sem_wait(&semDF);
    z = w * z;
    printf("z = %d\n", z);
    sem_post(&semFG);
}

void *processG()
{
    sem_wait(&semEG);
    sem_wait(&semFG);
    ans = y + z;
    printf("ans = %d\n", ans);
}

int main()
{
    pthread_t A, B, C, D, E, F, G;

    printf("nhap x1: ");
    scanf("%d", &x1);
    printf("nhap x2: ");
    scanf("%d", &x2);
    printf("nhap x3: ");
    scanf("%d", &x3);
    printf("nhap x4: ");
    scanf("%d", &x4);
    printf("nhap x5: ");
    scanf("%d", &x5);
    printf("nhap x6: ");
    scanf("%d", &x6);

    sem_init(&semAE, 0, 0);
    sem_init(&semAF, 0, 0);
    sem_init(&semBC, 0, 0);

```

```

int main()
{
    pthread_t A, B, C, D, E, F, G;

    printf("nhap x1:");
    scanf("%d", &x1);
    printf("nhap x2: ");
    scanf("%d", &x2);
    printf("nhap x3: ");
    scanf("%d", &x3);
    printf("nhap x4: ");
    scanf("%d", &x4);
    printf("nhap x5: ");
    scanf("%d", &x5);
    printf("nhap x6: ");
    scanf("%d", &x6);

    sem_init(&semAE, 0, 0);
    sem_init(&semAF, 0, 0);
    sem_init(&semBC, 0, 0);
    sem_init(&semBD, 0, 0);
    sem_init(&semCE, 0, 0);
    sem_init(&semDF, 0, 0);
    sem_init(&semEG, 0, 0);
    sem_init(&semFG, 0, 0);

    pthread_create(&A, NULL, &processA, NULL);
    pthread_create(&B, NULL, &processB, NULL);
    pthread_create(&C, NULL, &processC, NULL);
    pthread_create(&D, NULL, &processD, NULL);
    pthread_create(&E, NULL, &processE, NULL);
    pthread_create(&F, NULL, &processF, NULL);
    pthread_create(&G, NULL, &processG, NULL);

    while(1);
}

```


Chạy chương trình:

```
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ vim bai5.c
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ gcc bai5.c -o bai5 -lpthread -lrt
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ ./bai5
nhap x1: 1
nhap x2: 1
nhap x3: 1
nhap x4: 1
nhap x5: 1
nhap x6: 1
w = 1
v = 1
y = 1
z = 1
y = 1
z = 1
ans = 2
^C
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ ./bai5
nhap x1: 1
nhap x2: 2
nhap x3: 3
nhap x4: 4
nhap x5: 5
nhap x6: 6
w = 2
v = 12
y = 60
z = 72
y = 120
z = 144
ans = 264
^C
tuan_20522109@tuan-VirtualBox:~/Desktop/lab5$ ./bai5
nhap x1: 0
nhap x2: 2
nhap x3: 4
nhap x4: 3
nhap x5: 10
nhap x6: 6
w = 0
v = 12
y = 120
z = 72
y = 0
z = 0
ans = 0
```

