

Họ và tên: Nguyễn Nguyên Ngọc Anh

Mã số sinh viên: 22520058

Lớp: IT007.O11.1

HỆ ĐIỀU HÀNH BÁO CÁO LAB 5

CHECKLIST

5.5. BÀI TẬP THỰC HÀNH

	BT 1	BT 2	BT 3	BT 4
Trình bày cách làm	Có <input type="checkbox"/>	Có	Có	Có
Chụp hình minh chứng	Có	Có	Có	Có
Giải thích kết quả	Có	Có	Có	Có

5.6. BÀI TẬP ÔN TẬP

	BT 1
Trình bày cách làm	Có
Chụp hình minh chứng	Có
Giải thích kết quả	Có

Tư chấm điểm:

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*

<Tên nhóm>_LAB5.pdf

5.5. BÀI TẬP THỰC HÀNH

- 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]**

Trả lời...

- Cách làm :

```
/*#####  
# University of Information Technology #  
# IT007 Operating System  
#  
# Nguyen Nguyen Ngoc Anh, 22520058 #  
# File: 5_5_1.c #  
#####*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <unistd.h>  
  
sem_t sem;  
int sells = 0, products = 0;  
void *PROCESSA()  
{  
    while (1)  
    {  
        sem_wait(&sem);  
        sells++;  
        printf("sells = %d\n", sells);  
        sleep(2);  
    }  
}  
  
void *PROCESSB()  
{  
    while (1)  
    {  
        if (products <= sells + (22520058%1000) + 10)
```

```
    {
        products++;
        printf("products = %d\n", products);
        sem_post(&sem);
        sleep(1);
    }
}

void main()
{
    sem_init(&sem, 0, 0);
    pthread_t th1, th2;
    pthread_create(&th1, NULL, &PROCESSA, NULL);
    pthread_create(&th2, NULL, &PROCESSB, NULL);
    while (1);
}
```

- Chụp hình minh chứng :

```
nguyennguyennhocanh-22520058@DESKTOP-7R66M1N:~/Lab5$ gcc 5_5_1.c -o 5_5_1 -pthread
nguyennguyennhocanh-22520058@DESKTOP-7R66M1N:~/Lab5$ ./5_5_1
products = 1
sells = 1
products = 2
sells = 2
products = 3
products = 4
sells = 3
products = 5
products = 6
sells = 4
products = 7
products = 8
sells = 5
products = 9
products = 10
sells = 6
products = 11
products = 12
sells = 7
products = 13
products = 14
sells = 8
products = 15
products = 16
sells = 9
```

- Giải thích kết quả:

Trong PROCESSA, mỗi lần lặp, nó tăng biến sells lên 1 và in ra giá trị sells. Sau đó, nó dừng 2 giây (sleep(2)) trước khi lặp lại.

Trong PROCESSB, mỗi lần chu kỳ lặp, nó kiểm tra điều kiện if với $products \leq sells + (22520058 \% 1000) + 10$. Nếu điều kiện này đúng, nó tăng biến products lên 1, in ra giá trị products, và tăng giá trị của sem sử dụng `sem_post(&sem)` để báo hiệu cho PROCESSA rằng đã có sản phẩm mới. Sau đó, nó dừng 1 giây (sleep(1)) trước khi lặp lại.

Khi chạy chương trình này, hai luồng sẽ chạy song song. PROCESSB sẽ kiểm tra và tăng giá trị products dựa trên điều kiện và sau đó thông báo cho PROCESSA thông qua `sem_post`. PROCESSA sẽ tiếp tục tăng giá trị sells và in ra giá trị này. Đồng thời, việc

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

sleep giữa các lần tăng giá trị này sẽ khiến chương trình dừng lại trong một khoảng thời gian nhất định trước khi tiếp tục.

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.

Trả lời...

- Cách làm :

```
/*#####  
# University of Information Technology #  
# IT007 Operating System  
#  
# Nguyen Nguyen Ngoc Anh, 22520058 #  
# File: 5_5_2.c #  
#####*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <pthread.h>  
#include <semaphore.h>
```

```
#include <unistd.h>

sem_t sem1, sem2;
int n;
int i = 0;
static int dem = 0;
int a[1000000];
void *PROCESS1()
{
    while (1)
    {
        if (dem < n)
        {
            a[i++] = rand() % (n - 1);
            dem++;
            printf("\n[PUSH] Number of elements in array a: %2d", dem);
        }
        int time_sleep = rand() % 2 + 1;
        sleep(time_sleep);
        sem_post(&sem1);
    }
}
void *PROCESS2()
{
    int j, b;
    while (1)
    {
        sem_wait(&sem1);
```

```
    if (dem == 0)
    {
        printf("\n[POP] Nothing in array a");
    }
    else
    {
        dem--;
        b = a[0];
        for (j = 0; j < dem; j++)
        {
            a[j] = a[j + 1];
        }
        printf("\n[POP] Number of elements in array a: %2d", dem);
    }

    int time_sleep = rand() % 2 + 1;
    sleep(time_sleep);
}
}
void main()
{
    sem_init(&sem1, 1, 0);
    sem_init(&sem2, 0, 0);
    printf("\nEnter n: ");
    scanf("%d", &n);
    pthread_t th1, th2;
    pthread_create(&th1, NULL, PROCESS1, NULL);
    pthread_create(&th2, NULL, PROCESS2, NULL);
```


PROCESS2 sẽ đợi cho tín hiệu từ sem1 (bằng sem_wait) và sau đó kiểm tra xem có phần tử nào trong mảng không. Nếu có, nó sẽ loại bỏ phần tử đầu tiên ra khỏi mảng và giảm dem đi một. Sau đó, nó cũng ngủ một khoảng thời gian ngẫu nhiên trước khi tiếp tục.

Khi chạy chương trình với $n = 5$, nó sẽ tạo một mảng có thể chứa tối đa 5 phần tử và mô phỏng quá trình thêm và loại bỏ các phần tử từ mảng này.

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.

Trả lời...

- Cách làm :

```
/*#####
# University of Information Technology #
# IT007 Operating System
#
# Nguyen Nguyen Ngoc Anh, 22520058 #
# File: 5_5_3.c #
```

```
#####*/
```

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

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

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

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

```
#include <unistd.h>
```

```
int x = 0;
```

```
void* A()
```

```
{
```

```
while (1)
```

```
{
```

```
x = x + 1;
```

```
if (x == 20)
```

```
{
```

```
x = 0;
```

```
}
```

```
printf("PA: x = %d\n", x);
```

```
}
```

```
}
```

```
void* B()
```

```
{
```

```
while (1)
```

```
{
```

```
x = x + 1;
```

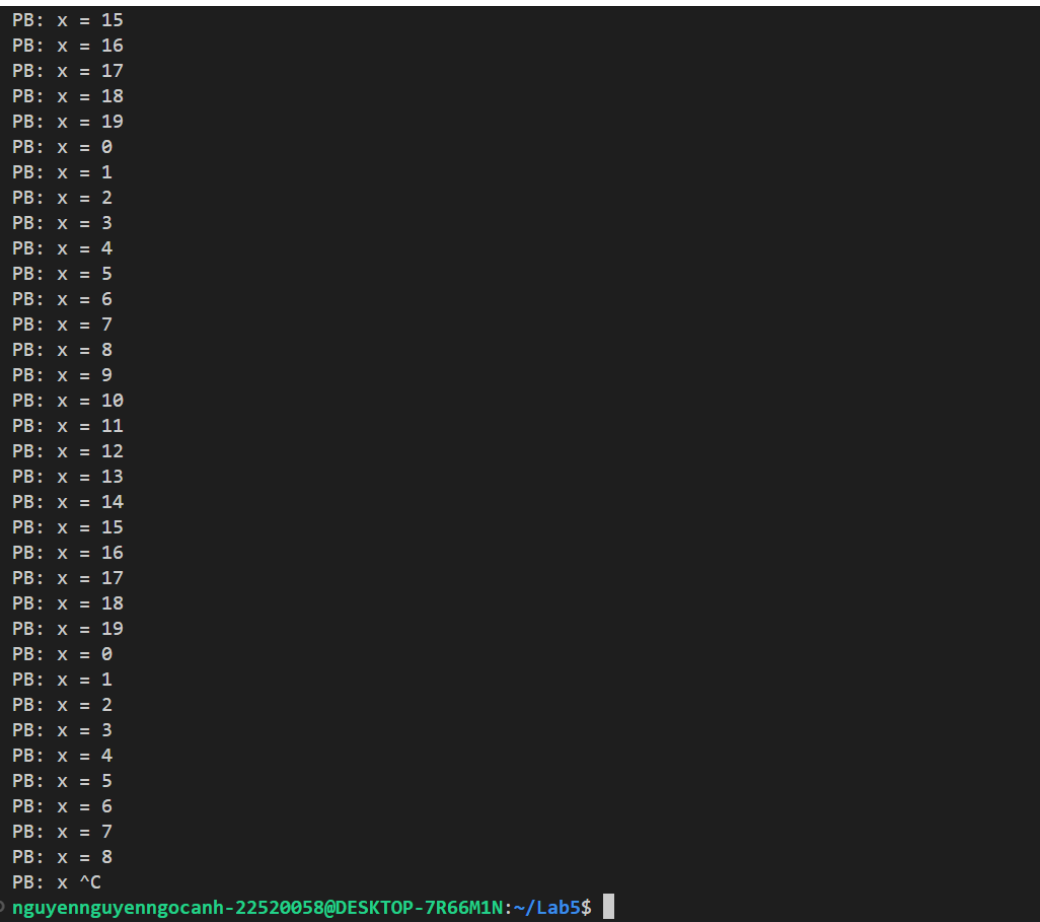
```
if (x == 20)
```

```
{
```

```
x = 0;
```

```
}  
printf("PB: x = %d\n", x);  
}  
sleep(1);  
}  
void main()  
{  
pthread_t th1, th2;  
pthread_create(&th1, NULL, &A, NULL);  
pthread_create(&th2, NULL, &B, NULL);  
while(1);  
}
```

- **Chụp hình minh chứng :**



```
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  
PB: x = 6  
PB: 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  
PB: x = 4  
PB: x = 5  
PB: x = 6  
PB: x = 7  
PB: x = 8  
PB: x = ^C
```

- `nguyennnguyenngocanh-22520058@DESKTOP-7R66M1N:~/Lab5$`

- **Giải thích kết quả:**

Trong hàm B. Dòng sleep(1); ở cuối hàm B sẽ không bao giờ được thực hiện vì nó đứng sau vòng lặp vô hạn while(1). Do đó, dòng lệnh này sẽ không bao giờ được thực thi.

Khi bạn chạy chương trình này, hai luồng A và B sẽ cùng chạy song song. Cả hai đều thực hiện việc tăng giá trị của x lên và kiểm tra nếu x đạt giá trị 20 thì gán x = 0.

Tuy nhiên, do lỗi trong hàm B (với dòng sleep(1); đứng sau vòng lặp vô hạn), chương trình sẽ không bao giờ ngủ và sẽ không bao giờ đến được dòng lệnh này. Kết quả là chương trình sẽ không dừng lại và hai luồng sẽ tiếp tục thực thi vô hạn, in ra giá trị của x sau mỗi lần tăng.

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

Trả lời...

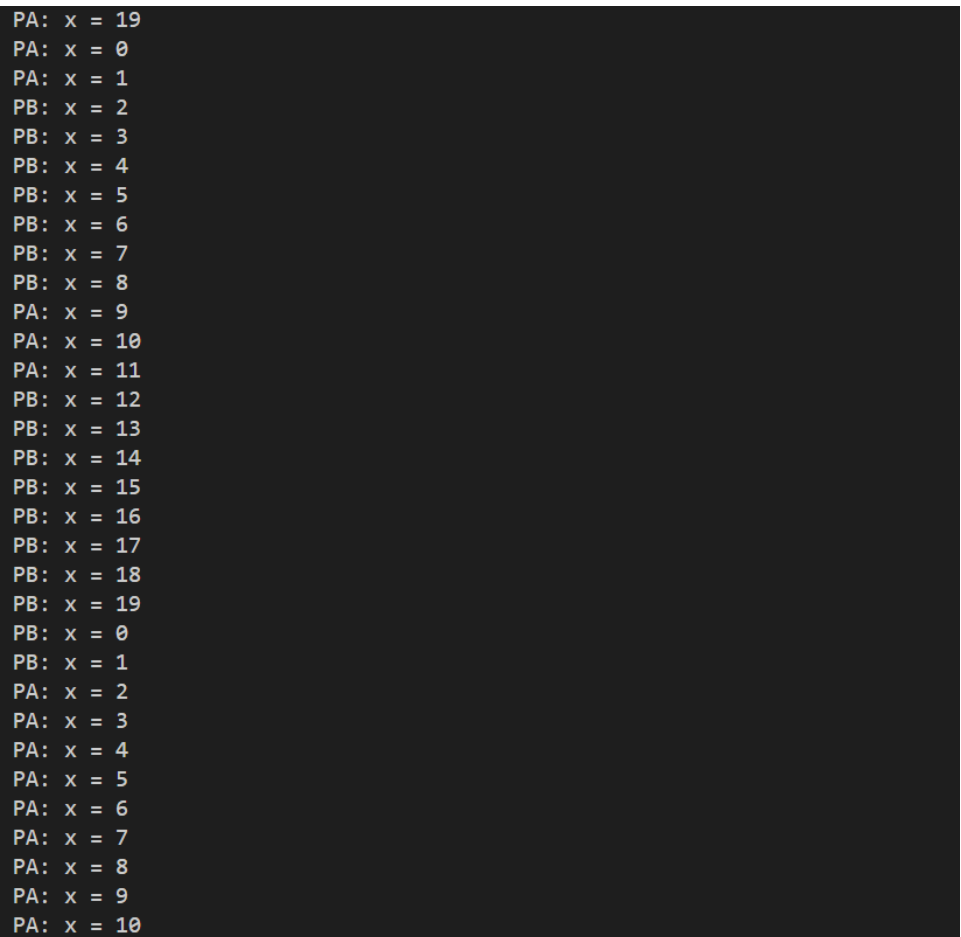
- **Cách làm :**

```
/*#####  
# University of Information Technology #  
# IT007 Operating System  
#  
# Nguyen Nguyen Ngoc Anh, 22520058 #  
# File: 5_5_4.c #  
#####*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <unistd.h>  
  
int x = 0;
```

```
pthread_mutex_t mutex;
void *PROCESS1()
{
    while (1)
    {
        pthread_mutex_lock(&mutex);
        x++;
        if (x == 20)
        {
            x = 0;
        }
        printf("PA: x = %d\n", x);
        pthread_mutex_unlock(&mutex);
    }
}
void *PROCESS2()
{
    while (1)
    {
        pthread_mutex_lock(&mutex);
        x++;
        if (x == 20)
        {
            x = 0;
        }
        printf("PB: x = %d\n", x);
        pthread_mutex_unlock(&mutex);
    }
}
```

```
void main()
{
    pthread_mutex_init(&mutex, NULL);
    pthread_t th1, th2;
    pthread_create(&th1, NULL, &PROCESS1, NULL);
    pthread_create(&th2, NULL, &PROCESS2, NULL);
    while (1);
}
```

- **Chụp hình minh chứng :**



```
PA: x = 19
PA: x = 0
PA: x = 1
PB: x = 2
PB: x = 3
PB: x = 4
PB: x = 5
PB: x = 6
PB: x = 7
PB: x = 8
PA: x = 9
PA: x = 10
PA: 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
PA: x = 2
PA: x = 3
PA: x = 4
PA: x = 5
PA: x = 6
PA: x = 7
PA: x = 8
PA: x = 9
PA: x = 10
```

- **Giải thích kết quả:**

Mục tiêu của việc sử dụng `pthread_mutex` là đảm bảo rằng chỉ một luồng có thể truy cập vào biến `x` tại một thời điểm. Khi một luồng khóa mutex bằng `pthread_mutex_lock`, nó sẽ

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

chờ cho đến khi nó có thể khóa mutex đó, đảm bảo rằng không có luồng nào khác có thể thực hiện lệnh nằm trong khoảng giữa `pthread_mutex_lock` và `pthread_mutex_unlock`.

Với mỗi lần lặp, cả hai hàm `PROCESS1` và `PROCESS2` đều khóa mutex trước khi thực hiện các thao tác trên biến `x`. Sau đó, sau khi thực hiện xong, họ mở khóa mutex bằng `pthread_mutex_unlock`.

Khi bạn chạy chương trình này, bạn sẽ thấy rằng giá trị của `x` được in ra màn hình sẽ tăng lên từ 0 đến 19, sau đó trở về 0 và tiếp tục lặp lại quá trình này. Sự thay đổi giữa `PA` và `PB` sẽ phụ thuộc vào việc nào trong hai luồng khóa mutex trước và tiếp tục thực hiện lệnh. Điều này thể hiện rằng việc sử dụng mutex đã đồng bộ hóa truy cập vào biến `x`, giúp tránh tình trạng đọc/ghi không đồng bộ mà có thể xảy ra nếu không sử dụng đồng bộ hóa.

5.6. 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) → (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

Trả lời...

- Cách làm :

```
/*#####  
# University of Information Technology #  
# IT007 Operating System  
#  
# Nguyen Nguyen Ngoc Anh, 22520058 #  
# File: 5_6.c #  
#####*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>
```

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

sem_t p1_5, p1_6, p2_3, p2_4, p3_5, p4_6, p5_7, p6_7;
int x1 = 1;
int x2 = 2;
int x3 = 3;
int x4 = 4;
int x5 = 5;
int x6 = 6;
int w, v, z, y, x;
int ans = 0;
void* PROCESS1()
{
    w = x1 * x2;
    printf("w = %d\n", w);
    sem_post(&p1_5);
    sem_post(&p1_6);
    sleep(1);
}
void* PROCESS2()
{
    v = x3 * x4;
    printf("v = %d\n", v);
    sem_post(&p2_3);
    sem_post(&p2_4);
    sleep(1);
}
void* PROCESS3()
```

```
{
sem_wait(&p2_3);
printf("y = %d\n", y);
y = v * x5;
sem_post(&p3_5);
sleep(1);
}

void *PROCESS4()
{
sem_wait(&p2_4);
printf("z = %d\n", z);
z = v * x6;
sem_post(&p4_6);
sleep(1);
}

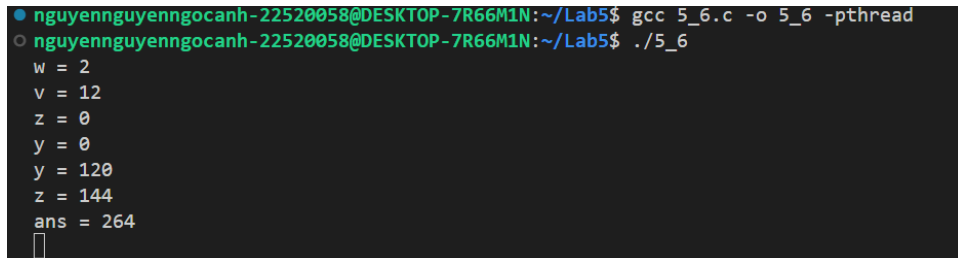
void *PROCESS5()
{
sem_wait(&p1_5);
sem_wait(&p3_5);
y = w * y;
printf("y = %d\n", y);
sem_post(&p5_7);
sleep(1);
}

void *PROCESS6()
{
sem_wait(&p1_6);
sem_wait(&p4_6);
z = w * z;
```

```
printf("z = %d\n", z);
sem_post(&p6_7);
sleep(1);
}
void* PROCESS7()
{
sem_wait(&p5_7);
sem_wait(&p6_7);
ans = y + z;
printf("ans = %d\n", ans);
sleep(1);
}
void main()
{
sem_init(&p1_5, 0, 1);
sem_init(&p1_6, 0, 0);
sem_init(&p2_3, 0, 0);
sem_init(&p2_4, 0, 0);
sem_init(&p3_5, 0, 0);
sem_init(&p4_6, 0, 0);
sem_init(&p5_7, 0, 0);
sem_init(&p6_7, 0, 0);
pthread_t th1, th2, th3, th4, th5, th6, th7;
pthread_create(&th1, NULL, &PROCESS1, NULL);
pthread_create(&th2, NULL, &PROCESS2, NULL);
pthread_create(&th3, NULL, &PROCESS3, NULL);
pthread_create(&th4, NULL, &PROCESS4, NULL);
pthread_create(&th5, NULL, &PROCESS5, NULL);
pthread_create(&th6, NULL, &PROCESS6, NULL);
```

```
pthread_create(&th7, NULL, &PROCESS7, NULL);  
while (1);  
}
```

- **Chụp hình minh chứng :**



```
nguyennnguyennngocanh-22520058@DESKTOP-7R66M1N:~/Lab5$ gcc 5_6.c -o 5_6 -pthread  
nguyennnguyennngocanh-22520058@DESKTOP-7R66M1N:~/Lab5$ ./5_6  
w = 2  
v = 12  
z = 0  
y = 0  
y = 120  
z = 144  
ans = 264  
█
```

- **Giải thích kết quả:**

PROCESS1 tính giá trị của w bằng tích của x1 và x2, in giá trị này ra màn hình và báo hiệu hoàn thành công việc thông qua p1_5 và p1_6.

PROCESS2 tính giá trị của v bằng tích của x3 và x4, in giá trị này ra màn hình và báo hiệu hoàn thành công việc thông qua p2_3 và p2_4.

PROCESS3 và PROCESS4 chờ tín hiệu từ PROCESS2 thông qua p2_3 và p2_4 rồi thực hiện tính toán tương ứng.

PROCESS5 và PROCESS6 chờ tín hiệu từ PROCESS1 thông qua p1_5 và p1_6 cùng với tín hiệu từ PROCESS3 và PROCESS4 để thực hiện tính toán.

PROCESS7 chờ tín hiệu từ PROCESS5 và PROCESS6 để thực hiện phép tính cuối cùng.

Mỗi hàm cũng ngủ 1 giây sau khi hoàn thành công việc của mình, nhưng vì vòng lặp vô hạn trong hàm main(), chương trình sẽ tiếp tục chạy mãi mãi sau khi tất cả các luồng đã hoàn thành công việc.

Khi chạy chương trình này, bạn sẽ thấy các thông điệp in ra màn hình theo thứ tự phù hợp với việc thực thi từng phép tính và các tín hiệu qua semaphore. Đồng thời, giá trị ans cuối cùng sẽ được in ra màn hình sau khi tất cả các phép tính đã hoàn thành.