

Name: Ngô Phúc Danh

ID: 21521924

Class: IT007.N11.1

OPERATING SYSTEM LAB 5'S REPORT

SUMMARY

Task		Status	Page
	Ex 1	Hoàn thành	2
	Ex 2	Hoàn thành	3
	Ex 3	Hoàn thành	5
	Ex 4	Hoàn thành	7
	Ex 5.5 (bonus)	Hoàn thành	9

Self-scores: 8

**Note: Export file to PDF and name the file by following format:
LAB1 – 21521924.pdf*

Câu1. Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau:
 $sells \leq products \leq sells + [2 \text{ số cuối của MSSV} + 10]$.

- **Hướng thực hiện:**
 - + Khởi tạo 2 biến `sells` và `products` có giá trị ban đầu đều bằng 0.
 - + Code 2 hàm `process sells` và `process products`, cài đặt các hàm `sem_wait` và `sem_post` cụ thể như sau.
 - + `Sem_wait` để chờ `products++` sao cho $\geq sells$ và `products` chỉ được tăng thêm giá trị sao cho $\leq sells + [24 + 10]$.
- **Hình ảnh minh họa:**



```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
sem_t sem;
int sells = 0, products = 0;
void *ProcessSells()
{
    while(1) {
        sem_wait(&sem);
        sells++;
        printf("Sells = %d\n",sells);
    }
}
void *ProcessProducts()
{
    while(1) {
        if (products <= sells + 24 + 10) {
            products++;
            printf("products = %d\n",products);
            sem_post(&sem);
        }
    }
}
void main() {
    sem_init(&sem,0,0);
    pthread_t process1, process2;
    pthread_create(&process1,NULL,&ProcessSells,NULL);
    pthread_create(&process2,NULL,&ProcessProducts,NULL);
    while(1);
}
```

Hình 1.1 Source code câu 1.

```

phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ vim bai5.4.1.c
phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ gcc bai5.4.1.c -o bai5.4.1 -lpthread -lrt
phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ ./bai5.4.1
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
products = 17
products = 18
Sells = 10
products = 19
products = 20
Sells = 11
products = 21
products = 22
Sells = 12
products = 23
products = 24
Sells = 13
products = 25
products = 26
Sells = 14

```

Hình 1.2 Kết quả khi chạy chương trình câu 1.

- Giải thích:

+ Khi đặt hàm sem_wait ở hàm processSells() nếu giá trị của sem = 0 thì tiến trình sẽ bị blocked cho đến khi giá trị semaphore > 0, khi này nếu giá trị của sem > 0 thì sẽ trừ đi 1 và chương trình tiếp tục chạy và xuất ra màn hình giá trị sells.

+ Khi đặt hàm sem_post trong hàm processProducts() thì một trong những tiến trình bị blocked bởi sem_wait sẽ được mở và sẵn sàng thực thi.

Câu 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.

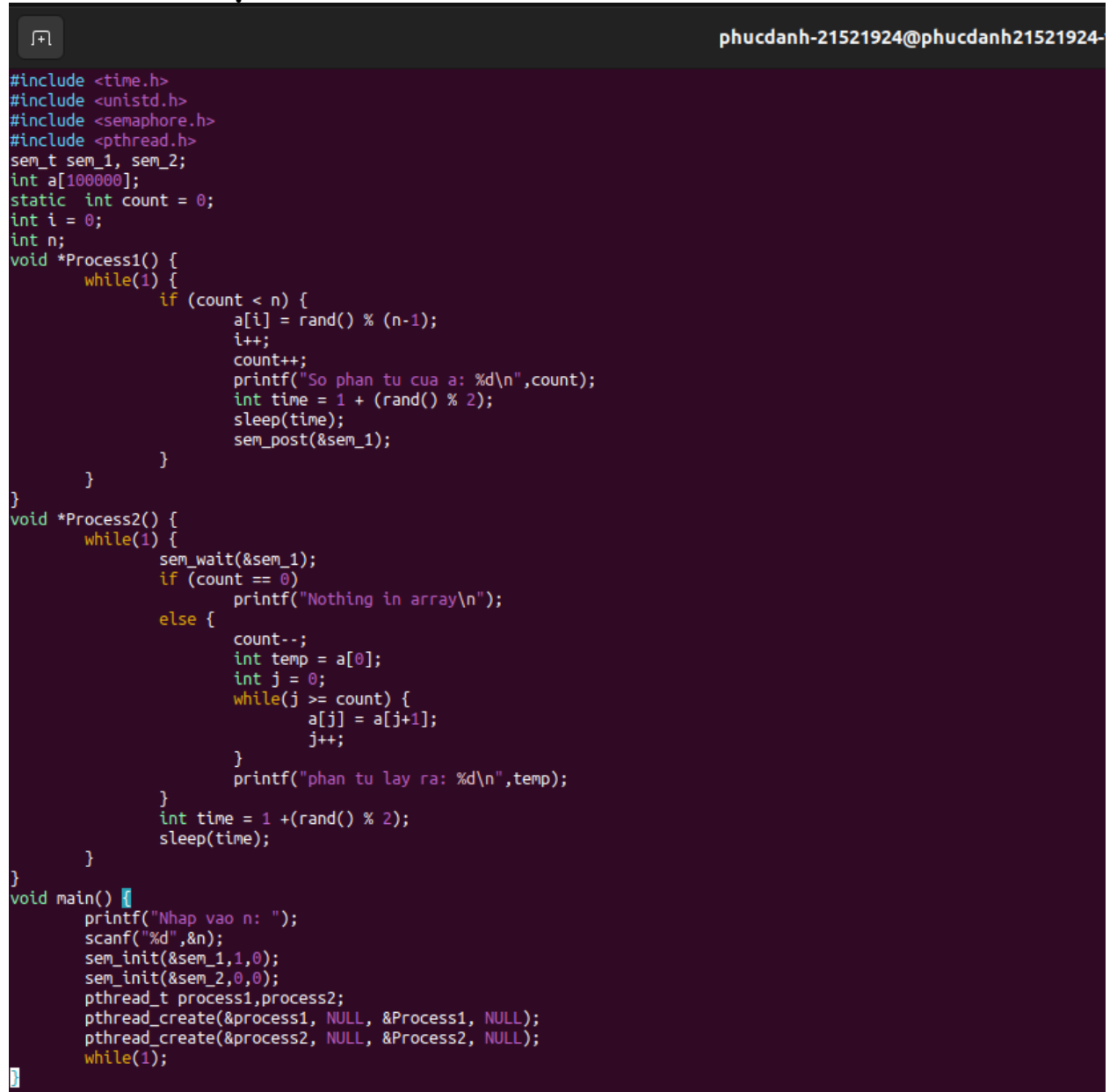
- Viết chương trình bao gồm 2 thread chạy song song: Một thread làm nhiệm vụ sinh ra một số nguyên ngẫu nhiên sau đó bỏ vào a. Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi thêm vào.
- Thread còn lại lấy ra một phần tử trong a (phần tử bất kỳ, phụ thuộc vào người lập trình). Sau đó đếm 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.

- **Hướng thực hiện:**

- + Tiến trình thêm vào mảng a: Random một giá trị vào mảng a sau đó tăng biến đếm thêm 1. Sau đó in giá trị biến đếm ra.
- + Tiến trình lấy một giá trị ra khỏi mảng a: chúng ta sẽ lấy phần tử đầu tiên sau đó gán các giá trị trước đó xuống 1 chỉ số. Nếu mảng không còn giá trị thì chúng ta in ra “Nothing in array”, nếu còn giá trị trong mảng a thì chúng ta in giá trị đầu tiên.

- **Hình ảnh minh họa:**



```
#include <time.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
sem_t sem_1, sem_2;
int a[100000];
static int count = 0;
int i = 0;
int n;
int j;
void *Process1() {
    while(1) {
        if (count < n) {
            a[i] = rand() % (n-1);
            i++;
            count++;
            printf("So phan tu cua a: %d\n",count);
            int time = 1 + (rand() % 2);
            sleep(time);
            sem_post(&sem_1);
        }
    }
}
void *Process2() {
    while(1) {
        sem_wait(&sem_1);
        if (count == 0)
            printf("Nothing in array\n");
        else {
            count--;
            int temp = a[0];
            int j = 0;
            while(j >= count) {
                a[j] = a[j+1];
                j++;
            }
            printf("phan tu lay ra: %d\n",temp);
        }
        int time = 1 + (rand() % 2);
        sleep(time);
    }
}
void main() {
    printf("Nhap vao n: ");
    scanf("%d",&n);
    sem_init(&sem_1,1,0);
    sem_init(&sem_2,0,0);
    pthread_t process1,process2;
    pthread_create(&process1, NULL, &Process1, NULL);
    pthread_create(&process2, NULL, &Process2, NULL);
    while(1);
}
```

Hình 2.1 Source code câu 2.

```

phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ vim bai5.4.2.c
phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ gcc bai5.4.2.c -o bai5.4.2 -lpthread -lrt
phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ ./bai5.4.2
Nhap vao n: 6
So phan tu cua a: 1
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 2
phan tu lay ra: 3
So phan tu cua a: 3
phan tu lay ra: 3
So phan tu cua a: 3
phan tu lay ra: 3
So phan tu cua a: 3
phan tu lay ra: 3
So phan tu cua a: 3
phan tu lay ra: 3
So phan tu cua a: 3
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3
So phan tu cua a: 3
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3
So phan tu cua a: 4
phan tu lay ra: 3

```

Hình 2.2 Kết quả khi chạy chương trình câu 2.

- **Giải thích:**

- + Đặt hàm `sem_post()` trong khi cài đặt hàm `process1()` giúp mở khóa tiến trình đang bị block bởi `sem_wait()`, về nội dung code thì chúng ta kiểm tra xem nếu biến đếm $\geq n$ thì chúng ta không thêm vào mảng `a` và ngược lại. Sau khi thêm, chúng ta in số phần tử trong mảng `a` ra màn hình.
- + Đặt hàm `sem_wait()` trong khi cài đặt hàm `process2()` giúp blocked, nếu giá trị `sem = 0` thì tiếp tục bị blocked còn nếu >0 thì chương trình thực thi bình thường và `sem` trừ đi 1.

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

int x = 0;	
PROCESS A	PROCESS B
processA() {	processB() {

12

<pre>while(1){ x = x + 1; if (x == 20) x = 0; print(x); }</pre>	<pre>while(1){ x = x + 1; if (x == 20) x = 0; print(x); }</pre>
---	---

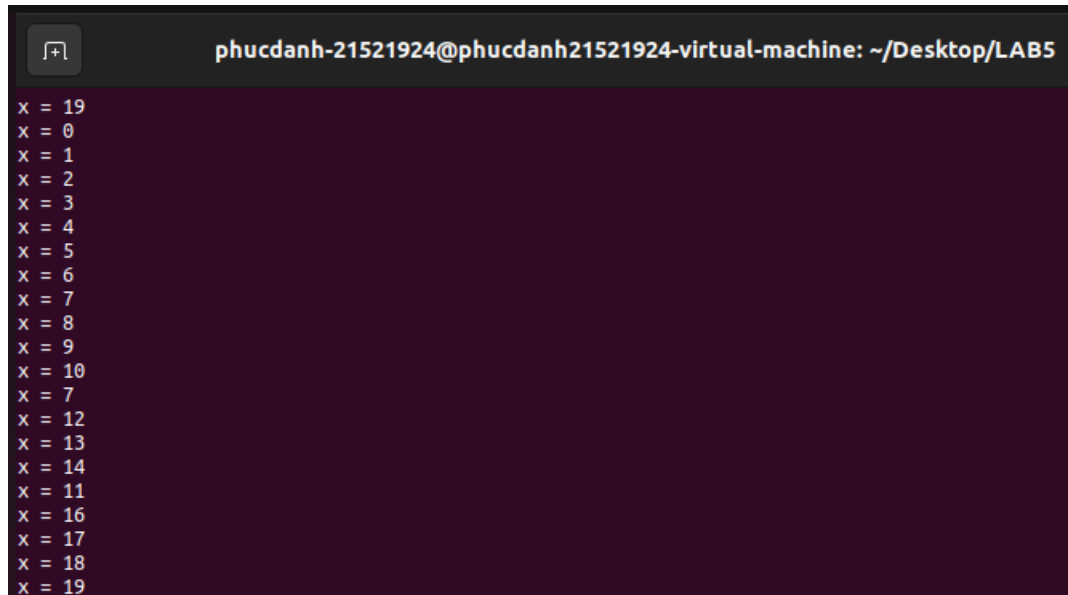
Hình 3.1 2 process A và B chạy song song.

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

- **Hướng thực hiện:** Code lại đoạn code trên ubuntu.
- **Hình ảnh minh họa:**

```
phucdanh-21521924@phucdanh21521924-virtual-machine:
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>
int x = 0;
void *PROCESSA() {
    while(1) {
        x = x + 1;
        if (x==20)
            x = 0;
        printf("x = %d\n",x);
    }
}
void *PROCESSB() {
    while(1) {
        x = x + 1;
        if (x==20)
            x = 0;
        printf("x = %d\n",x);
    }
}
void main() {
    pthread_t process1, process2;
    pthread_create(&process1, NULL, &PROCESSA, NULL);
    pthread_create(&process2, NULL, &PROCESSB, NULL);
    while(1);
}
```

Hình 3.1 Source code câu 3.



```
phucdanh-21521924@phucdanh21521924-virtual-machine: ~/Desktop/LAB5
x = 19
x = 0
x = 1
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
x = 10
x = 7
x = 12
x = 13
x = 14
x = 11
x = 16
x = 17
x = 18
x = 19
```

Hình 3.2 Kết quả khi chạy chương trình câu 3.

- **Giải thích:** Ta thấy vấn đề khi $x = 10$ và sau đó $x = 7$ mà không phải bằng 11. Lý do là vì khi 2 tiến trình được chạy song song khi 1 tiến trình thực hiện việc ++ và lưu giá trị thì hết thời gian được cấp phép cho tiến trình đó thì nó sẽ lấy giá trị của tiến trình còn lại. ta tạm gọi giá trị tính toán của 10 là thuộc tiến trình A và 7 của tiến trình B, khi đó vấn đề xảy ra khi in x ra 7 thuộc tiến trình B.

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

- **Hướng thực hiện:** Chúng ta thực hiện việc đặt `pthread_mutex_lock` và `pthread_mutex_unlock` ở từng `PROCESSA` và `PROCESSB` để nếu tại một hàm đang bị khóa thì nó sẽ vẫn bị khóa lại cho tới khi mutex được mở ra và việc này lặp đi lặp lại liên tục.
- **Hình ảnh minh họa:**

```
phucdanh-21521924@phucdanh21521924-  
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.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);  
}
```

Hình 4.1 Source code câu 4.


```
phucdanh-21521924@phucdanh21521924
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
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
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
```

Hình 4.2 Kết quả khi chạy chương trình câu 4.

- **Giải thích:**

- + Đối với hàm PROCESSA khi đặt pthread_mutex_lock. Được tham chiếu bởi con trỏ *mutex, nếu nó đã bị khóa trước đó thì tiếp tục bị khóa cho tới khi mutex được mở ra. Sau khi được mở và thực thi code trong PROCESSA thì mở khóa mutex được tham chiếu bởi con trỏ *mutex và sau khi mở thì các tiến trình khác có quyền tranh chấp khóa mutex.
- + Đối với hàm PROCESSB cũng tương tự như trên.

Câu 5.5 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:

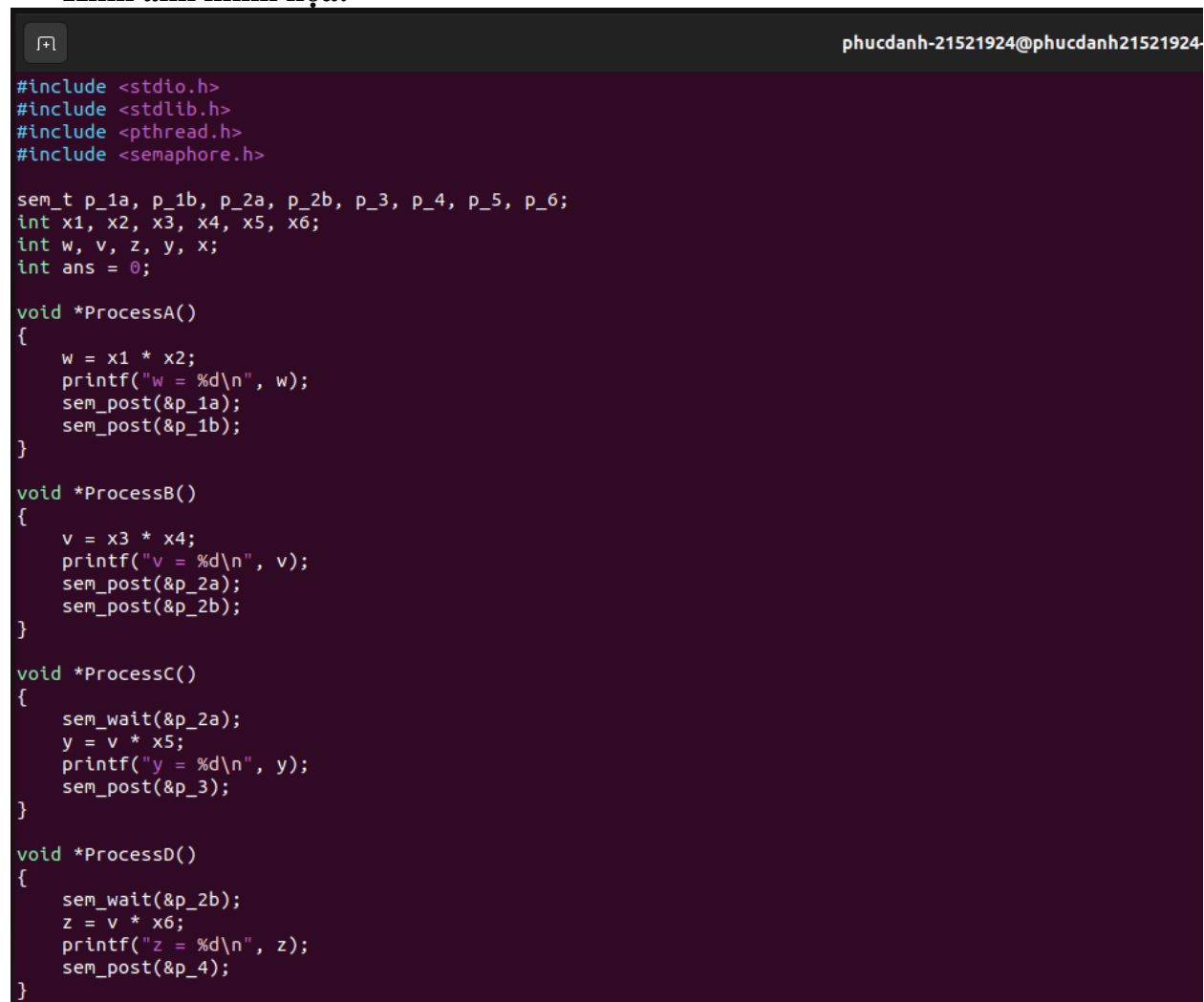
(c), (d) chỉ được thực hiện sau khi v được tính

(e) chỉ được thực hiện sau khi w và y được tính

(g) chỉ được thực hiện sau khi y và z được tính

- **Hướng thực hiện:** cài đặt từng phép tính thành các hàm tương đương với các tiến trình từ (a->g) \Leftrightarrow (1-7). Sau đó tạo các biến sem và cài đặt sem_wait và sem_post sao cho thỏa yêu cầu đề.

- **Hình ảnh minh họa:**



```

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

sem_t p_1a, p_1b, p_2a, p_2b, p_3, p_4, p_5, p_6;
int x1, x2, x3, x4, x5, x6;
int w, v, z, y, x;
int ans = 0;

void *ProcessA()
{
    w = x1 * x2;
    printf("w = %d\n", w);
    sem_post(&p_1a);
    sem_post(&p_1b);
}

void *ProcessB()
{
    v = x3 * x4;
    printf("v = %d\n", v);
    sem_post(&p_2a);
    sem_post(&p_2b);
}

void *ProcessC()
{
    sem_wait(&p_2a);
    y = v * x5;
    printf("y = %d\n", y);
    sem_post(&p_3);
}

void *ProcessD()
{
    sem_wait(&p_2b);
    z = v * x6;
    printf("z = %d\n", z);
    sem_post(&p_4);
}
  
```

Hình 5.5.1 Source code câu 5.5.

```
phucdanh-21521924@phucdanh21521924-

printf("z = %d\n", z);
sem_post(&p_4);
}

void *ProcessE()
{
    sem_wait(&p_1a);
    sem_wait(&p_3);
    y = w * y;
    printf("y = %d\n", y);
    sem_post(&p_5);
}

void *ProcessF()
{
    sem_wait(&p_1a);
    sem_wait(&p_4);
    z = w * z;
    printf("z = %d\n", z);
    sem_post(&p_6);
}

void *ProcessG()
{
    sem_wait(&p_5);
    sem_wait(&p_6);
    ans = y + z;
    printf("ans = %d\n", ans);
}

void main()
{
    printf("Nhap vo x1, x2, x3, x4, x5, x6: ");
    scanf("%d %d %d %d %d %d", &x1, &x2, &x3, &x4, &x5, &x6);
    sem_init(&p_1a, 0, 1);
    sem_init(&p_1b, 0, 0);
    sem_init(&p_2a, 0, 0);
    sem_init(&p_2b, 0, 0);
    sem_init(&p_3, 0, 0);
    sem_init(&p_4, 0, 0);
    sem_init(&p_5, 0, 0);
    sem_init(&p_6, 0, 0);
    pthread_t thread1, thread2, thread3, thread4, thread5, thread6, thread7;
    pthread_create(&thread1, NULL, &ProcessA, NULL);
    pthread_create(&thread2, NULL, &ProcessB, NULL);
    pthread_create(&thread3, NULL, &ProcessC, NULL);
    pthread_create(&thread4, NULL, &ProcessD, NULL);
    pthread_create(&thread5, NULL, &ProcessE, NULL);
    pthread_create(&thread6, NULL, &ProcessF, NULL);
    pthread_create(&thread7, NULL, &ProcessG, NULL);
}

Show Applications
```

Hình 5.5.2 Source code câu 5.5 (tt).

```
phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ vim bai5.5.c
phucdanh-21521924@phucdanh21521924-virtual-machine:~/Desktop/LAB5$ ./bai5.5
Nhap vo x1, x2, x3, x4, x5, x6: 1 2 3 4 5 6
w = 2
v = 12
y = 60
z = 72
z = 144
y = 120
ans = 264
```

Hình 5.5.3 Kết quả khi chạy chương trình câu 5.5.

- **Giải thích:** Tiến trình a và b là 2 tiến trình chạy song song đầu tiên mà không cần điều kiện, tiến trình c và d cần chờ tiến trình b chạy xong vì vậy ta đặt `sem_wait()`, tiến trình e và f cần giá trị của tiến trình c và d sau khi tính toán nên ta đặt thêm `sem_post()` vào c và d. Tiến trình e và f cần chờ tiến trình a, b, c, d chạy xong nên

ta đặt `sem_wait()`. Cuối cùng với tiến trình tính toán `ans` thì ta đặt `sem_wait()` để chờ tiến trình `e` và `f` tính toán xong. Kết quả sau đó được xuất ra màn hình.

.....HẾT.....