|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **MSSV** | **Lớp** |
| Lại Quan Thiên | 22521385 | IT007.O211.1 |
| Đặng Đức Tài | 22521270 |
| Mai Nguyễn Nam Phương | 22521164 |
| Phùng Trần Thế Nam | 21522366 |

HỆ ĐIỀU HÀNH  
BÁO CÁO LAB 5

**CHECKLIST**

**5.5. BÀI TẬP THỰC HÀNH**

**Phân công:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BT 1** | **BT 2** | **BT 3** | **BT 4** | **BTOT** |
| **Lại Quan Thiên** |  |  |  |  |  |
| **Đặng Đức Tài** |  |  |  |  |  |
| **Mai Nguyễn Nam Phương** |  |  |  |  |  |
| **Phùng Trần Thế Nam** |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BT 1** | **BT 2** | **BT 3** | **BT 4** |
| **Trình bày cách làm** |  |  |  |  |
| **Chụp hình minh chứng** |  |  |  |  |
| **Giải thích kết quả** |  |  |  |  |

**5.6. BÀI TẬP ÔN TẬP**

|  |  |
| --- | --- |
|  | **BT 1** |
| **Trình bày cách làm** |  |
| **Chụp hình minh chứng** |  |
| **Giải thích kết quả** |  |

**Tự chấm điểm:** 10/10

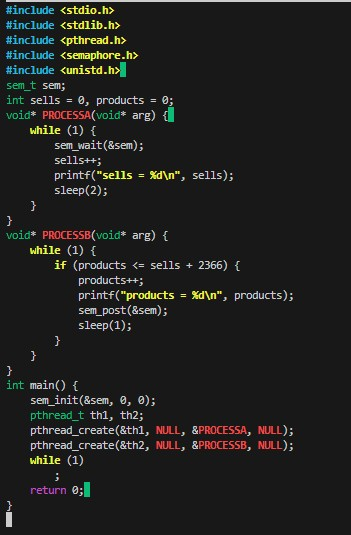
*\*Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:* ***<Tên nhóm>\_LAB5.pdf***

**5.5. BÀI TẬP THỰC HÀNH**

# 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]

# (MSSV: 21522366)

**\* Code:**



**\* Giải thích:**

**- Khai báo thư viện và biến:**

+ Chương trình bắt đầu bằng việc khai báo các thư viện cần thiết bao gồm stdio.h, stdlib.h, pthread.h, semaphore.h và unistd.h (để sử dụng hàm sleep).

+ Biến toàn cục sells và products được khai báo để đếm số lượng hàng bán được và số lượng sản phẩm đã được tạo ra.

**- Hàm PROCESSA:**

+ Hàm này chạy trong một luồng riêng biệt.

+ Trong vòng lặp vô hạn nó sử dụng sem\_wait để chờ đợi tín hiệu từ luồng PROCESSB.

+ Khi nhận được tín hiệu nó tăng biến sells lên 1 đơn vị và in ra giá trị của sells.

+ Sau đó chờ 2 giây trước khi lặp lại.

**- Hàm PROCESSB:**

+ Hàm này cũng chạy trong một luồng riêng biệt.

+ Trong vòng lặp vô hạn nó kiểm tra điều kiện products <= sells + 2366.

+ Nếu điều kiện này được thỏa mãn, nó tăng biến products lên 1 đơn vị in ra giá trị của products và gửi một tín hiệu tới semaphone sem bằng sem\_post.

+ Sau đó chờ 1 giây trước khi lặp lại.

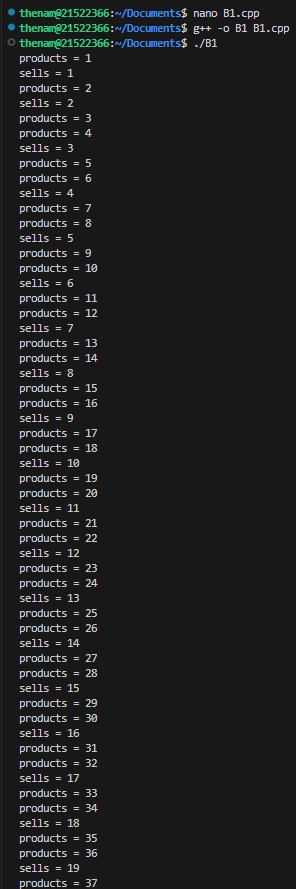
**- Hàm main:**

+ Trong hàm main, semaphone, và sem được khởi tạo với giá trị ban đầu là 0 bằng sem\_init.

+ Hai luồng mới được tạo bằng pthread\_create, một cho PROCESSA và một cho PROCESSB.

**=> Điều này tạo ra một môi trường mô phỏng việc bán hàng và sản xuất hàng, trong đó hai luồng hoạt động song song và tương tác với nhau thông qua một semaphone.**

**\* Màn hình kết quả:**



# 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:

* **Chương trình khi chưa được đồng bộ:**

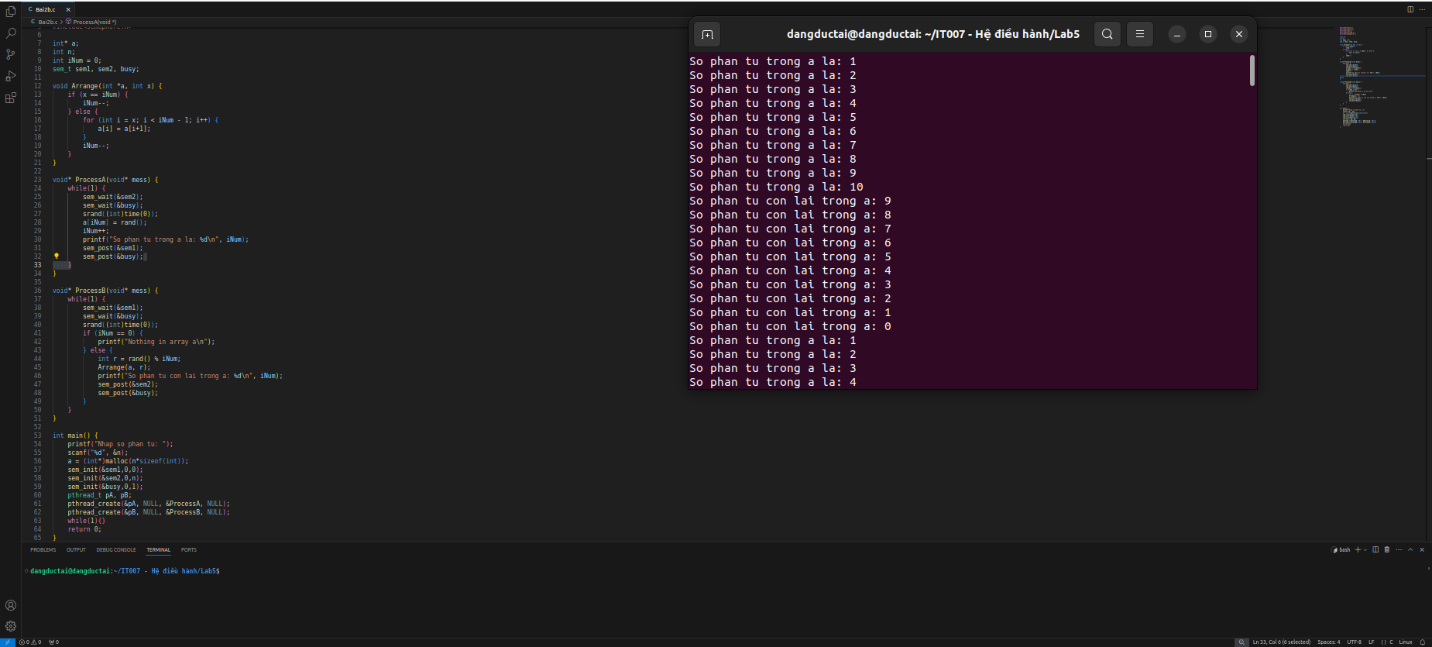
A screenshot of a computer

Description automatically generated

**\* Giải thích:** Hàm Arrange có chức năng sắp xếp lại hàm khi lấy ngẫu nhiên một phần tử trong mảng a ra. Vì chưa có semaphore nên chương trình chưa được đồng bộ. Xảy ra lỗi do B lấy ra kích thước của mảng A khi A chưa được vào chương trình.

=> Cần dùng semaphore để xử lí.

* **Chương trình sau khi được thêm semaphore:**



**\* Giải thích:**

- Biến sem1 đóng vai trò điều kiện đảm bảo rằng sẽ không thể lấy phần tử trong A khi không có phần tử nào.

- Biến sem2 đóng vai trò điều kiện đảm bảo rằng sẽ không thể thêm phần tử quá số lượng phần tử được cho phép (mỗi lần chỉ lấy 1 phần tử).

- Khi chương trình được nạp, giả sử ProcessB được nạp trước sẽ bị khóa bởi biến sem1 (khởi tạo = 0). Khi đó ProcessA được chạy, tiếp tục thêm vào phần tử có giá trị ngẫu nhiên vào a và tăng sem1 lên 1 (bởi hàm sem\_post) cho đến khi sem2=0 (được giảm bởi hàm sem\_wait). Khi đó ProcessB được chạy, lấy một phần tử ngẫu nhiên trong a ra và tăng sem2 và sem1 (bởi hàm sem\_post) cho đến khi sem1 = 0 (được giảm bởi hàm sem\_wait). Hai tiến trình này sẽ được lặp đi lặp lại trong 1 vòng lặp vô hạn.

# 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);  }  } | |

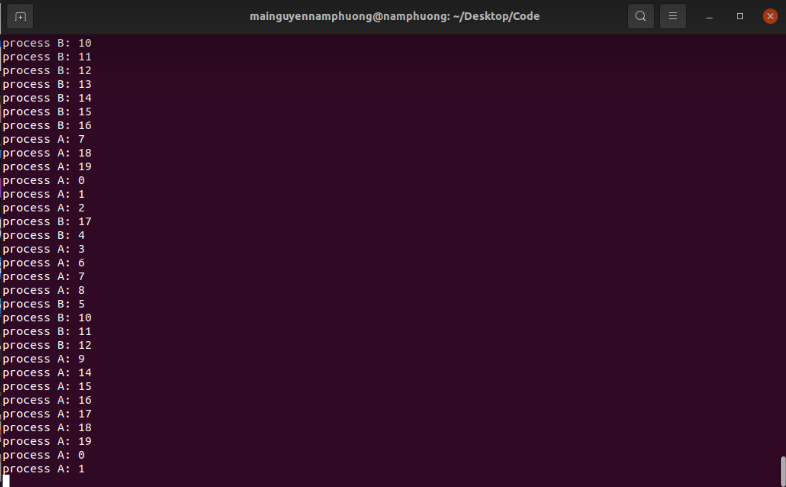
Hiện thực mô hình trên C trong hệ điều hành Linux và nhận xét kết quả.

**\* Source code:**

Ảnh có chứa văn bản, máy tính, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

**\* Kết quả:**



**\* Nhận xét:** Process thứ nhất chạy từ 1 đến 19 và khi đến if(x==20) thì sẽ quay lại giá trị 0. Và trong khi process 1 đang chuẩn bị đọc dữ liệu từ đĩa, thao tác đọc chưa hoàn tất thì bị process 2 ghi đè dữ liệu mới lên dữ liệu cũ. Nguyên nhân là do khi chạy song song 2 process thứ nhất và thứ hai, biến toàn cục x đều được cho cả hai process biến x gọi ra nhưng không có semaphore để báo rằng đây là hai tiến trình dùng chung biến x dẫn đến việc xảy ra đụng độ khi truy cập và xử lí biến chung như dòng trên. Từ đó dẫn đến kết quả sai với yêu cầu và chương trình chạy bất hợp lý.

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

**\* Source code:**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Mô tả được tạo tự động**

**\* Kết quả:**

**Ảnh có chứa ảnh chụp màn hình, văn bản, Phần mềm đa phương tiện, phần mềm

Mô tả được tạo tự động**

**\* Nhận xét:** Saukhi được đồng bộ với mutex thì ta đã sửa được lỗi trong bài 3, 2 tiến trình đã được báo rằng biến x cục bộ được sử dụng chungnên chương trình đã chạy đúng như yêu cầu

**5.6. BÀI TẬP ÔN TẬP**

# 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)

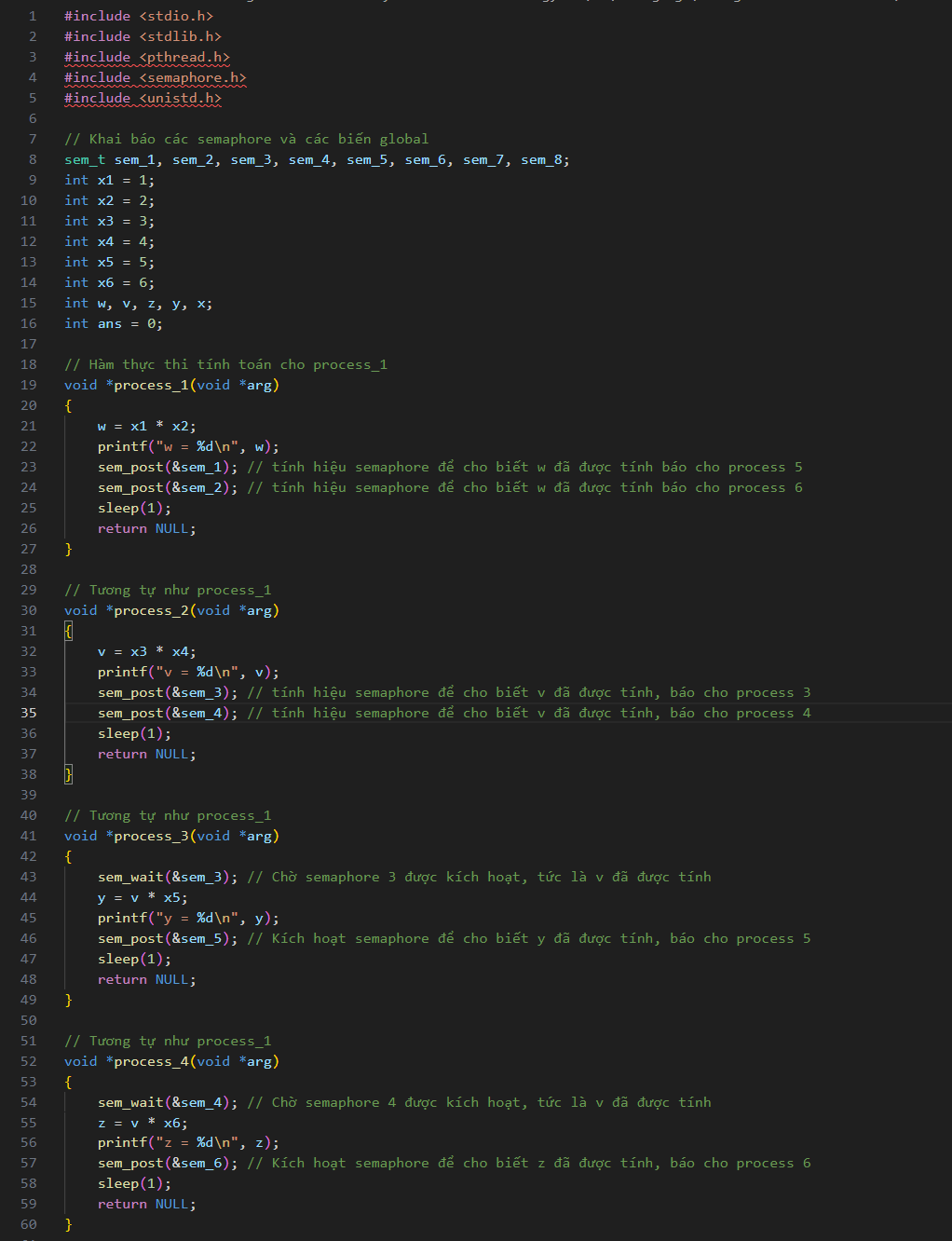
Giả sử các lệnh từ (a) à (g) nằm trên các thread chạy song song với nhau. Hãy lập trình mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự sau:

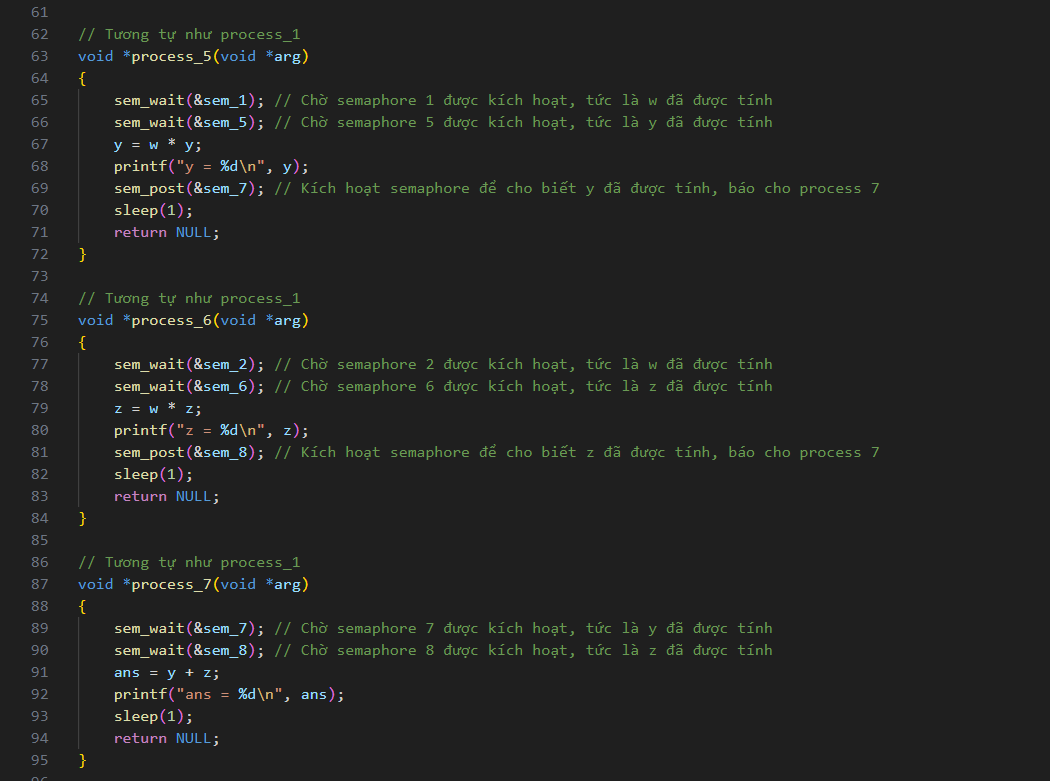
(c), (d) chỉ được thực hiện sau khi v được tính

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

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

\* Code:





A screen shot of a computer program

Description automatically generated

**\* Giải thích:**

- **Khởi tạo Semaphore:** Trong hàm main(), các semaphore được khởi tạo với giá trị ban đầu là 0. Semaphore được sử dụng để đồng bộ hóa các phép tính và đảm bảo rằng chúng được thực hiện theo đúng thứ tự.

- **Tạo các luồng:** Trong hàm main(), bảy luồng (thread) được tạo bằng cách gọi hàm

pthread\_create. Mỗi luồng sẽ thực hiện một phép tính nhất định. Cụ thể là:

+ process\_1 tính toán giá trị của biến w.

+ process\_2 tính toán giá trị của biến v.

+ process\_3 tính toán giá trị của biến y.

+ process\_4 tính toán giá trị của biến z.

+ process\_5 tính toán giá trị của biến y một lần nữa, sau khi đã có giá trị của w.

+ process\_6 tính toán giá trị của biến z một lần nữa, sau khi đã có giá trị của w.

+ process\_7 tính toán giá trị của biến ans, tức là tổng của y và z.

- **Đồng bộ hóa các phép tính:** Các semaphore được sử dụng để đảm bảo rằng các phép tính được thực hiện theo đúng thứ tự:

+ sem\_1 và sem\_2 cho biết rằng x1 \* x2 đã được tính toán và ghi vào w, gửi tính hiệu cho process\_5 và process\_6.

+ sem\_3 và sem\_4 cho biết rằng x3 \* x4 đã được tính toán và ghi vào v, gửi tính hiệu cho process\_3 và process\_4

+ sem\_5 cho biết rằng y đã được tính toán, gửi tính hiệu cho process\_5.

+ sem\_6 cho biết rằng z đã được tính toán, gửi tính hiệu cho process\_6.

+ sem\_7 và sem\_8 cho biết rằng cả y và z đã được tính toán, gửi tính hiệu cho process\_7.

- **In kết quả:** Khi tất cả các luồng đã hoàn thành, hàm main() in ra giá trị của biến ans, tức là kết quả của phép tính y + z.

