|  |
| --- |
| **Name: Vương Đinh Thanh Ngân**  **ID: 20521649**  **Class: ATCL2022** |

OPERATING SYSTEM  
LAB 1’S REPORT

**SUMMARY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | | **Status** | **Page** |
| Section 1.5 | Ex 5.3.3 | Done | 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| Ex 5.3.4 | Done | 9, 11, 12 |

**Self-scrores:**

*\*Note: Export file to* ***PDF*** *and name the file by following format:* ***LAB X – <Student ID>.pdf***

**5.3.3: Thực hiện câu hỏi chuẩn bị**

- Phân biệt các khái niệm chương trình (program), tiến trình (process) và tiểu trình (thread)

|  |  |  |
| --- | --- | --- |
| Program | Process | Thread |
| - Chương trình là là tập hợp các câu lệnh thực hiện một tác vụ cụ thể khi được máy tính thực thi. | - Tiến trình là một thực thể của một chương trình máy tính đang được thực thi bởi một hoặc nhiều luồng. | - Tiểu trình là đơn vị xử lý cơ bản trong hệ thống. Trong một tiến trình có thể có nhiều tiểu trình, nó xử lý tuần tự đoạn code của nó, sở hữu một con trỏ lệnh, một tập các thanh ghi và một vùng nhớ stack riêng và các tiểu trình cũng chia sẻ thời gian xử lý của processor như các tiến trình. |

- Sự tranh chấp xảy ra khi nào? cho ví dụ?

Sự tranh chấp xảy ra khi nhiều process truy xuất và thao tác đồng thời lên dữ liệu chia sẻ.

- Phân biệt sự khác nhau giữa 2 nhóm giải pháp: “busy waiting” và “sleep & wake up”. Liệt kê một số hệ điều hành sử dụng 2 nhóm giải pháp trên.

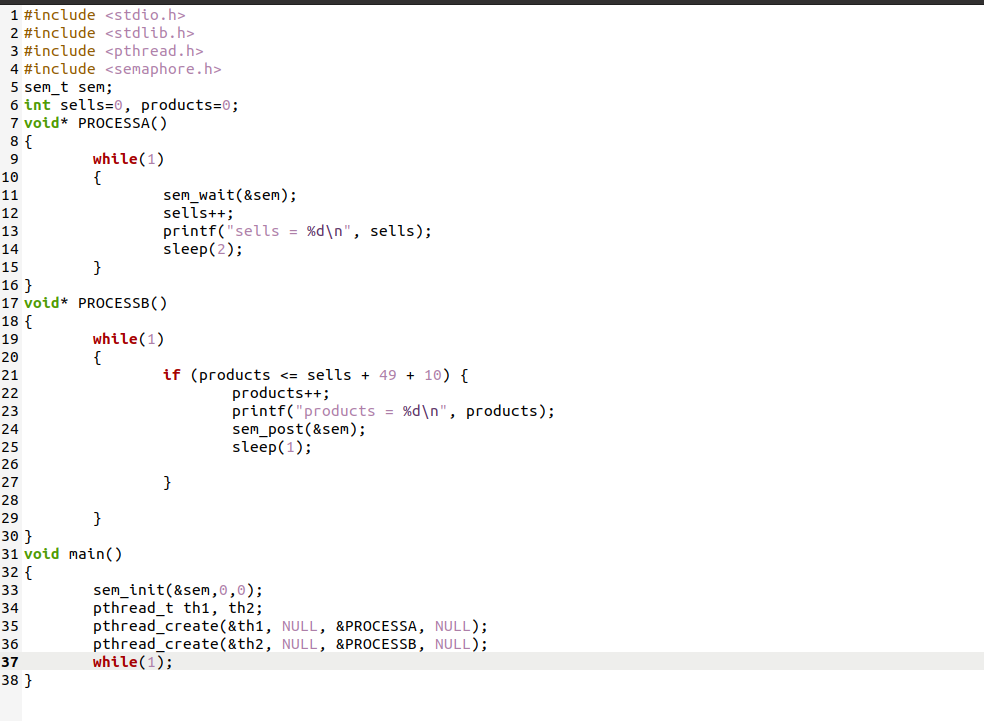
|  |  |
| --- | --- |
| Busy waiting | Sleep & wake up |
| - Sử dụng các biến cờ hiệu  - Sự dựng việc kiểm tra luân phiên  - Giải pháp của Peterson  - Cấm ngắt  - Các giải pháp busy waiting thì While (chưa có quyền) do nothing()  - Tiếp tục chờ đợi CPU trong khi vào miền găng, không đòi hỏi sự trợ giúp của hệ điều hành  - Dùng trên hệ điều hành: Linux, Unix | - Semaphore  - Monitor  - Message  - Các giải pháp sleep & wake up thì if(chưa có quyền) sleep ()  - Từ bỏ CPU khi chưa đc vào CS, cần sự hộ trở từ hệ điều hành  - Dùng trên hệ điều hành: Linux, Unix |

**5.4:Hướng dẫn thực hành**

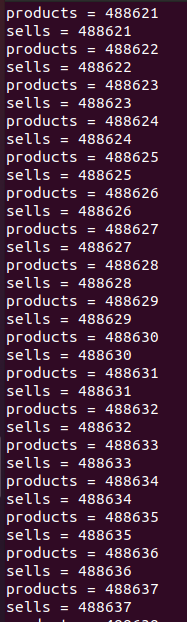
Bài 1:

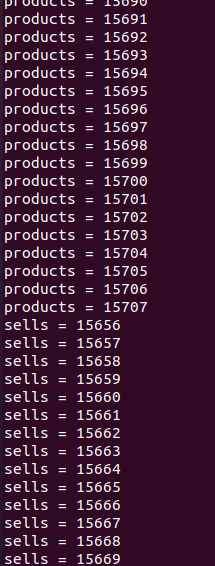
- Ta có 2 điều kiện : Sells <= Products và Products <= Sells +49 + 10

- Giả định Sells = Products = 0 ta có 0<=0<= 59



*Hình 5.1: Đoạn code chương trình mô hình trong ví dụ 5.3.1.2*





*Hình 5.2: Thực thi chương trình mô hình trong ví dụ 5.3.1.2*

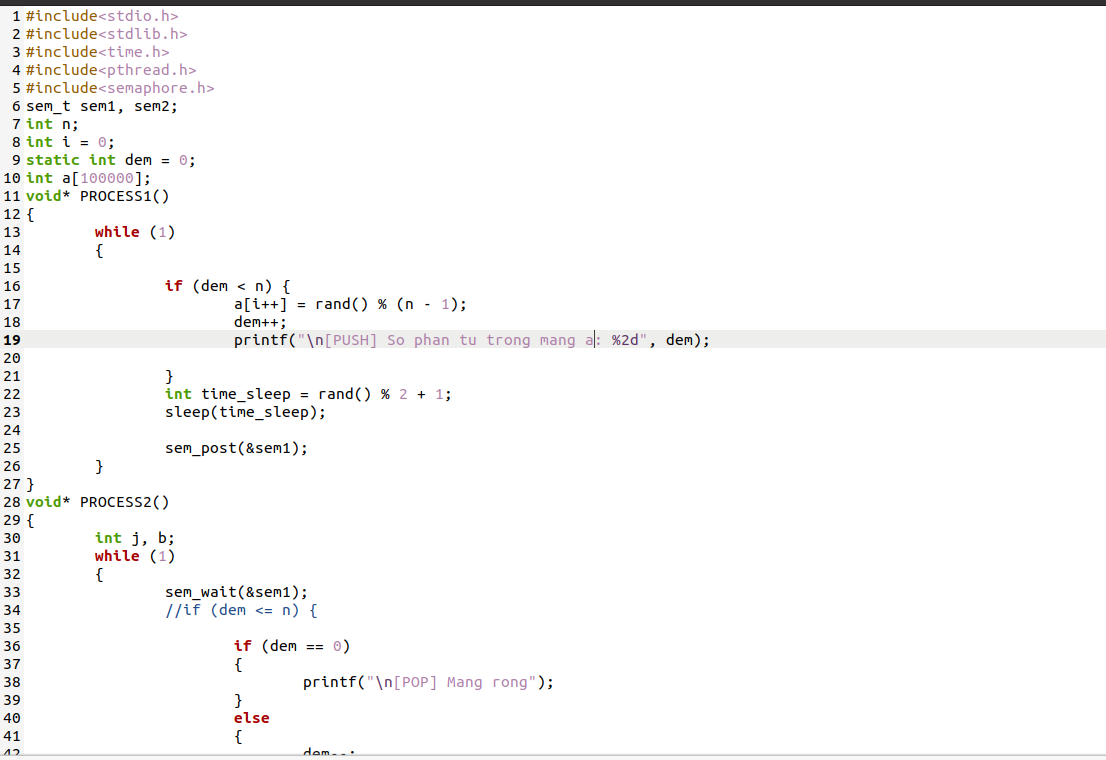
Cần 2 biến Semaphore để đồng bộ 2 tiến trình:

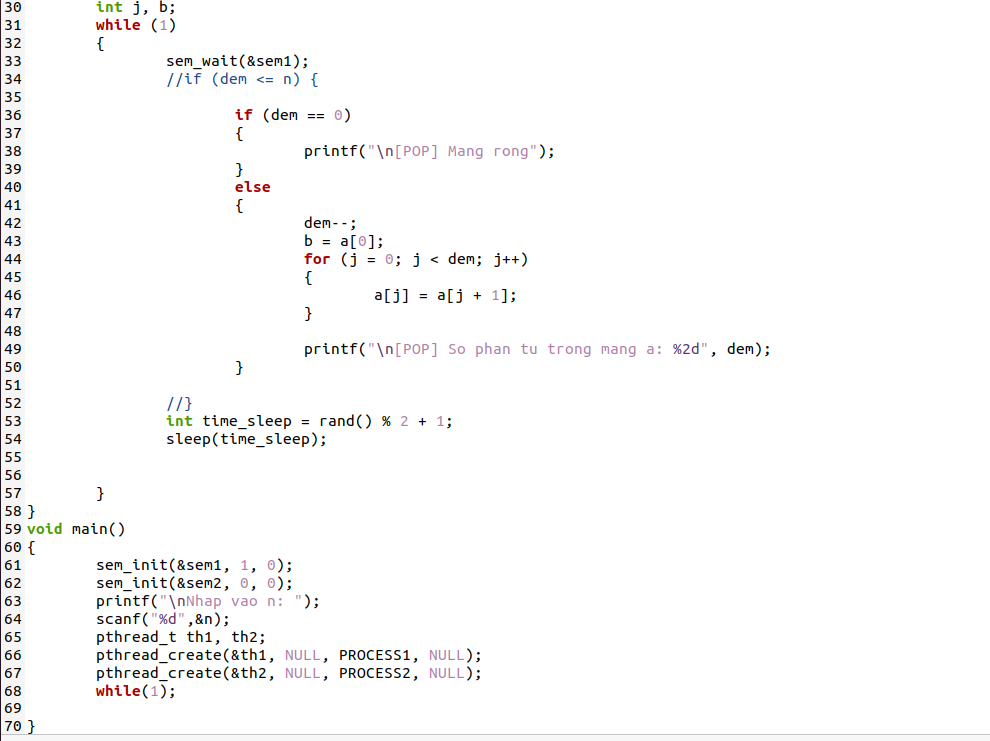
- Với ProcessA thực hiện viện bán (tức là Sells++) để + Sells cần kiểm tra điều kiện Sells <= Products vì vậy cần đặt hàm sem\_wait(&sem1) trước khi thực thi câu lệnh Sells++ để ProcessA bị chặn đầu tiên vì nếu ko chặn thì Sells++ sẽ là 1<=0 (vô lý) do đó tiến trình bị chặn <=> sem1 – 1 < 0 => sem1 < 1 => sem1 = 0 sau khi thực thi các câu lệnh xong cần cập nhật lại điều kiện Sells+59 cho đúng với hiện tại ta cần dòng lệnh cuối cùng là sem\_post(&sem2).

- Tương tự với Process B ta được sem2 =59 và đặt hàm sem\_wait(&sem2) trước khi thực hiện câu lệnh Products++ để kiểm tra xem Products <= Sells +59 và thực thi xong cần gọi lệnh sem\_post(&sem1) để tăng giá trị sem1 lên cập nhật điều kiện cho Product.

Bài 2:

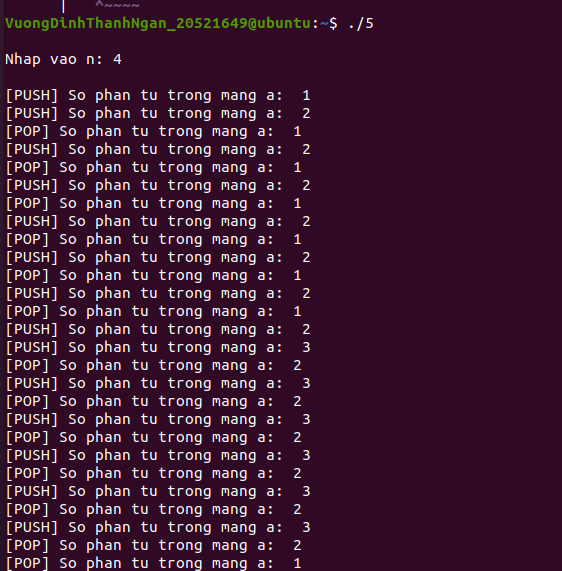
- Ý tưởng tạo mảng động kiểu vector A để dễ dàng quản lý và biến n là số phần tử tối đa nhập từ bàn phím và biến để hiện số phần tử hiện tại của vector A





*Hình 5.3: Đoạn code chương trình bao gồm 2 thread chạy song song*

- Cũng tương tự như bài trên vì tốc độ xử lý của hệ điều hành quá cao nên ta hầu như không thấy lỗi gì. Để thấy rõ hơn lỗi này em thử cho ProcessAdd bị crassh bằng dòng lệnh sleep() trước khi push vào vector.



*Hình 5.4: Thực thi chương trình bao gồm 2 thread chạy song song*

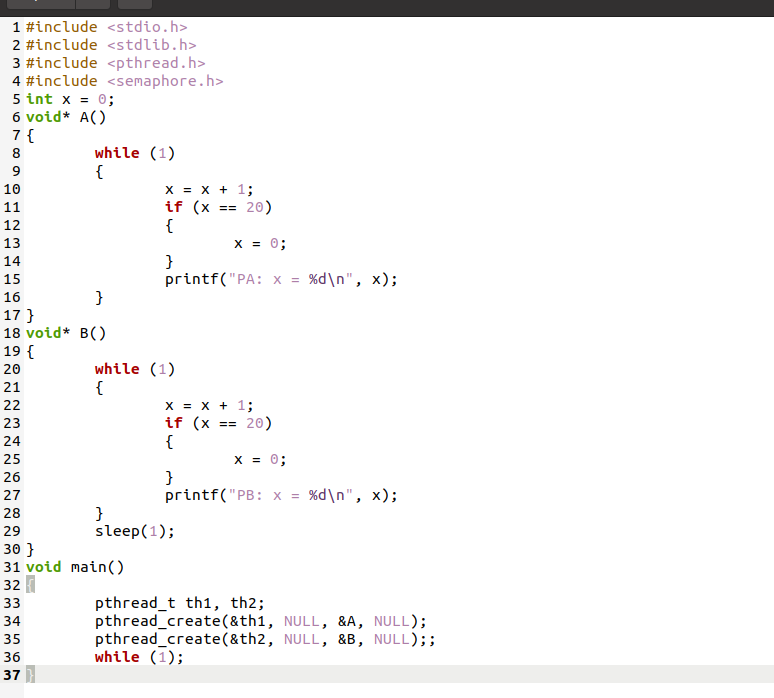
- Để giải quyết vấn đề này cần động bộ hóa 2 tiến trình trên.

- Điều kiện tiến trình Add thực thi: dem < n (cần 1 biến semaphore)

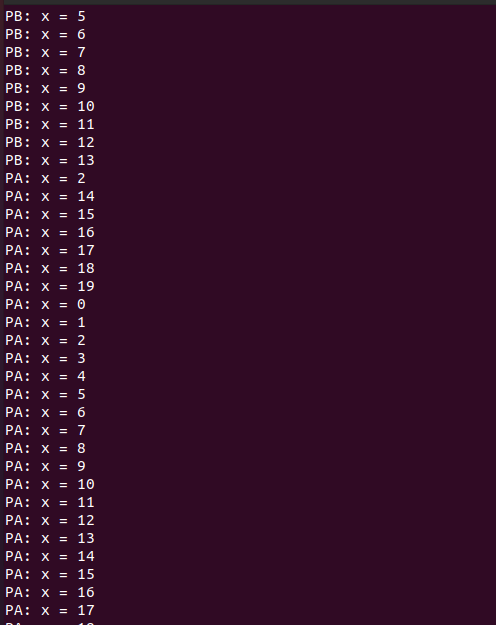
- Điều kiện tiến trình Sub thực thi: dem > 0 (cần 1 biến semaphore)

- Do 2 điều kiện trên ta cần khai báo 2 biến semaphore để đồng bộ hóa 2 tiến trình (sem1 và sem2)

Bài 3:



*Hình 5.5: Đoạn code process A và B chạy song song*



Hình 5.6: Thực thi chương trình process A và B chạy song song

Bài 4:

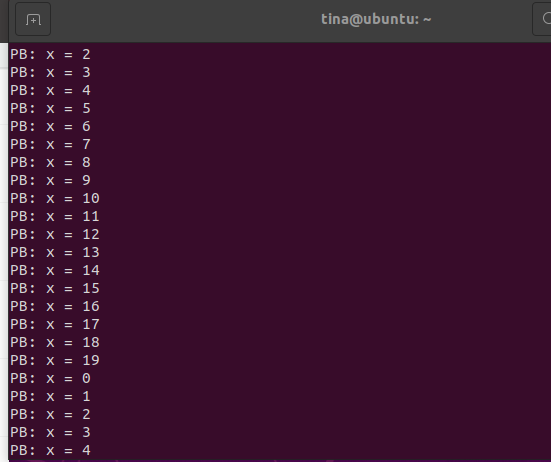


*Hình 5.6: Đoạn code đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình Bài 3.*

- Kết quả sẽ Xuất giá trị x của ProcessA từ 0 đến 19 cho đến khi nào hết time slice và ProcessB tương tự do dùng chung biến chia sẻ x.

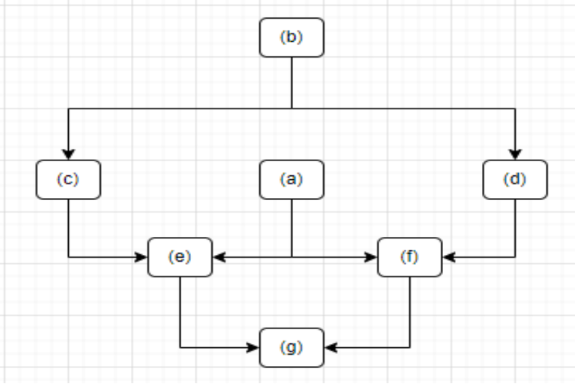
- Cần đặt vùng tranh chấp trong vòng while của 2 tiến trình được giới hạn bới 2 hàm pthread\_mutex\_lock và pthread\_mutex\_unlock cơ chế hoạt động như sau

- Tại một thời điểm Process gọi hàm lock sẽ khóa mutex lúc này nếu Process còn lại cùng đồng thời gọi hàm lock thì Process đó sẽ bị block chờ cho đến khi nào Process kia thực thi xong và gọi hàm unlock thì lúc này nó mới được phép thực thi do đó sẽ không bao giờ tồn tại trường hợp x có giá trị vượt quá 20.



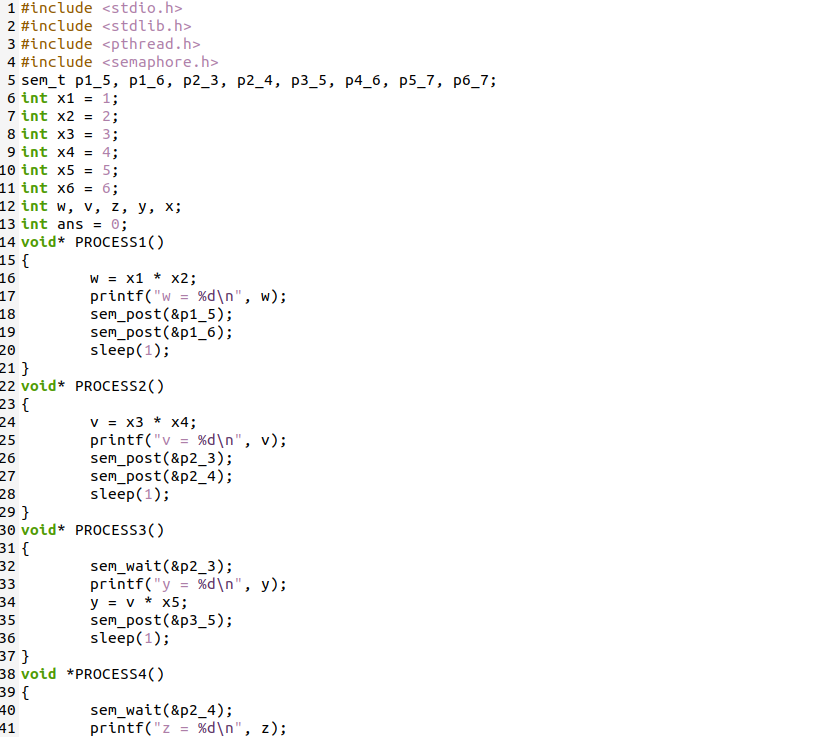
*Hình 5/8: Thực thi code đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình Bài 3.*

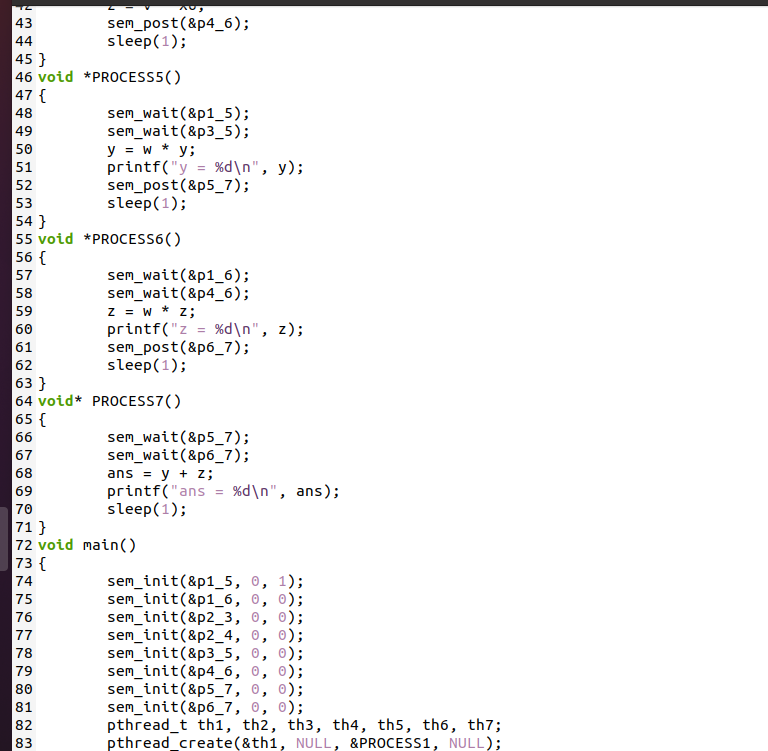
**5.5: Bài tập ôn tập**



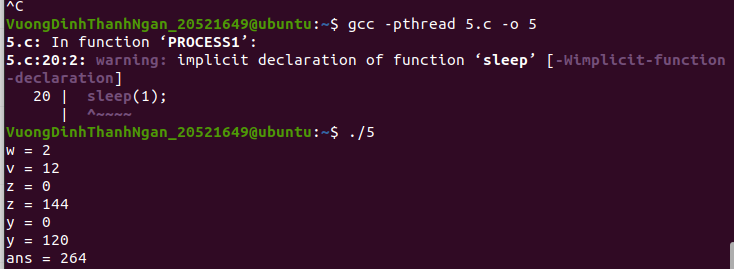
*Hình 5.9: Mô hình bài 5*

- Từ đó xây dựng được chương trình đồng bộ bằng 5 biến semaphore



**

*Hình 5.10: Đoạn code mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự*



*Hình 5.11: Thực thi code mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự*

- Khi Buid chương trình nhập x1,x2,x3,x4,x5,x6 lần lượt bằng 1,2,3,4,5,6 ta có kết quả ans đầu tiên bằng 264 là đúng