Name: Phan Trọng Tính

MSSV: 21522683

Class: IT007-N17.1

OPERATING SYSTEM

LAB 5'S REPORT

SUMMARY

Task		Status	Page
5.4	1	Hoàn thành	1
	2	Hoàn thành	3
	3	Hoàn thành	
	4	Hoàn thành	
5.5	1	Hoàn thành	
5.6	1	Hoàn thành	
	2	Hoàn thành	

Self-scrores:

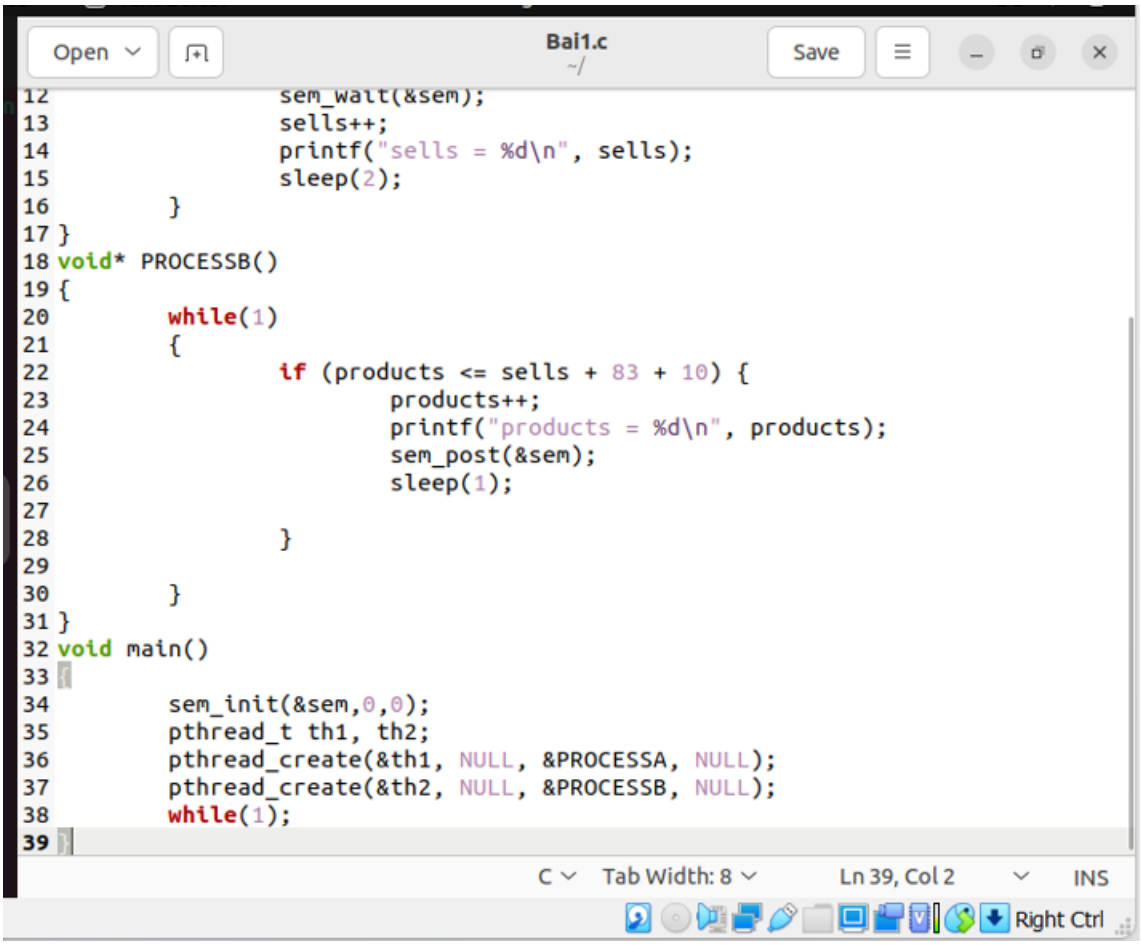
Note: Export file to **PDF and name the file by following format:*

LAB X – <Student ID>.pdf

5.4 Hướng dẫn 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 + [2 số cuối của MSSV + 10]**

- **Bước 1:** Tạo file Bai1.c (sử dụng câu lệnh $ gedit Bai1.c)
- **Bước 2:** Nhập nội dung cho file Bai1.c sau đó lưu lại

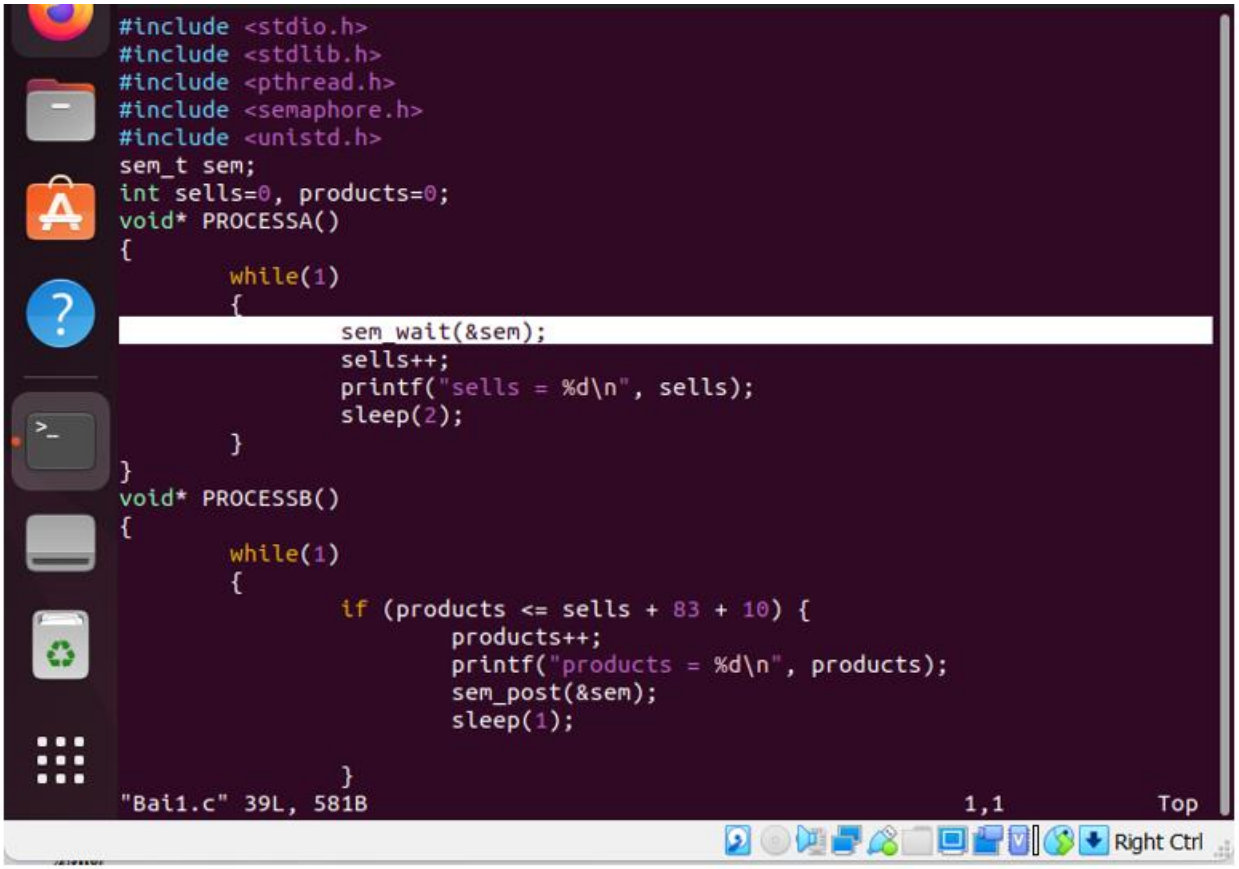


- **Bước 3:** Kiểm tra file Bai1.c đã được tạo thành công hay chưa (sử dụng câu lệnh $ ls)

  - Nếu chưa có file Bai1.c thì quay lại bước 1
  - Nếu đã có file Bai1.c thì tiếp tục đến bước 4

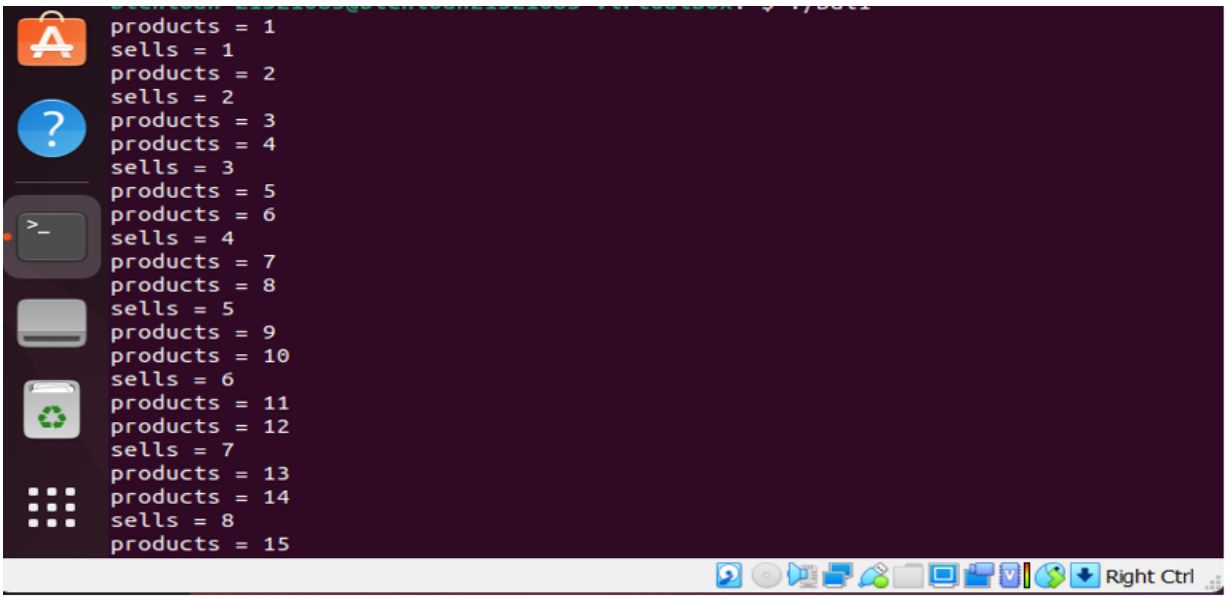
  File Bai1.c đã có trong danh sách => Tiếp tục đến bước 4

- **Bước 4:** Mở file Bai1.c bằng vim (sử dụng câu lệnh $ vim Bai1.c)



- **Bước 5:** Thực thi file Bai1.c (sử dụng câu lệnh $ gcc -pthread Bai1.c -o Bai1)

- **Bước 6:** Thực thi file Bai1 (sử dụng câu lệnh $ ./Bai1)

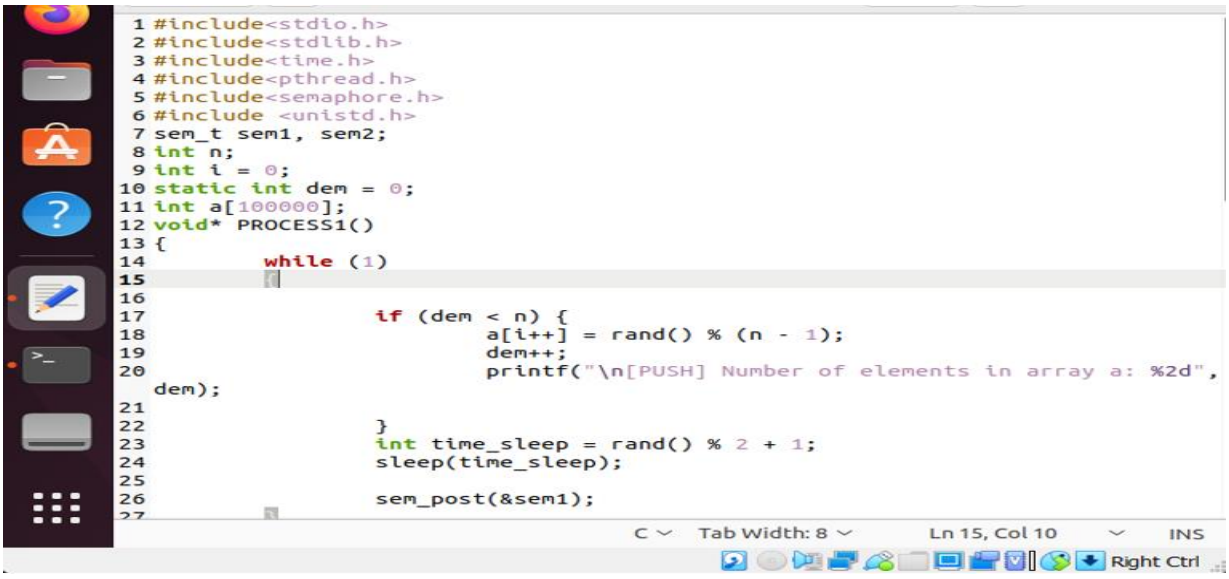


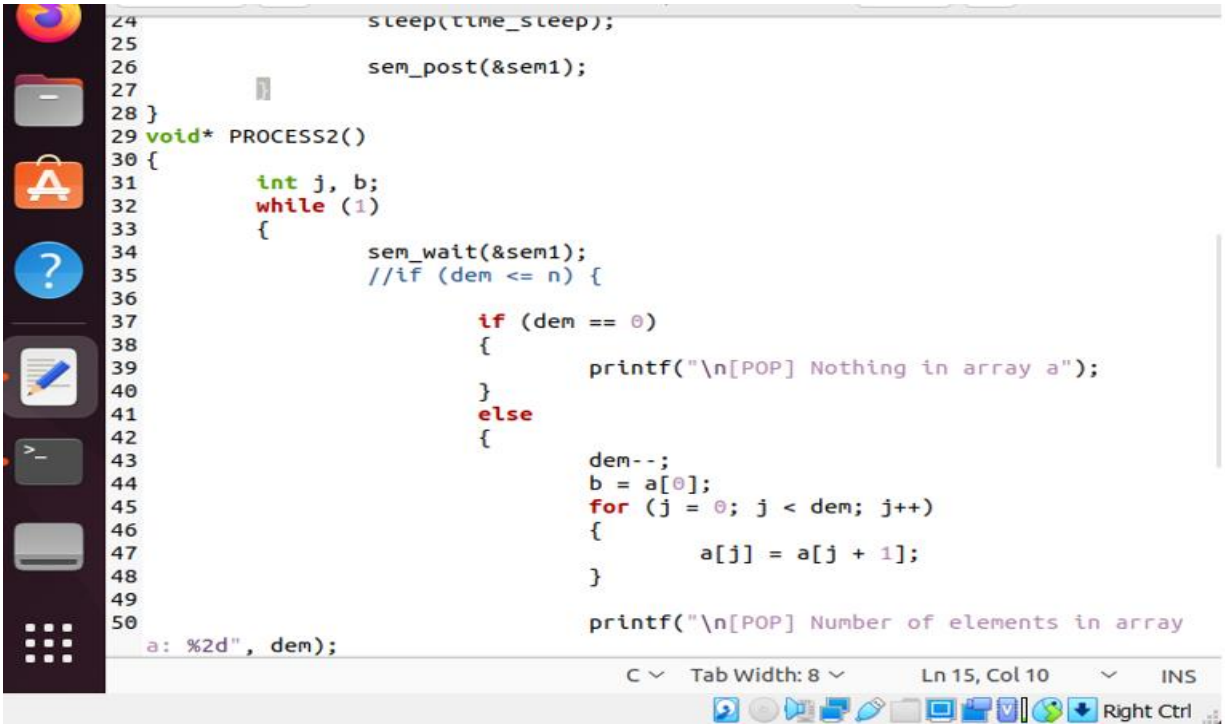
2. Cho một mảng a được khai báo như một mảng số nguyên có thể chứa n phần tử, a được khai báo như một biến toàn cục. Viết chương trình bao gồm 2 thread chạy song song:

- 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.

- **Bước 1:** Tạo file Bai2.c (sử dụng câu lệnh $ gedit Bai2.c)
- **Bước 2:** Nhập nội dung cho file Bai2.c sau đó lưu lại

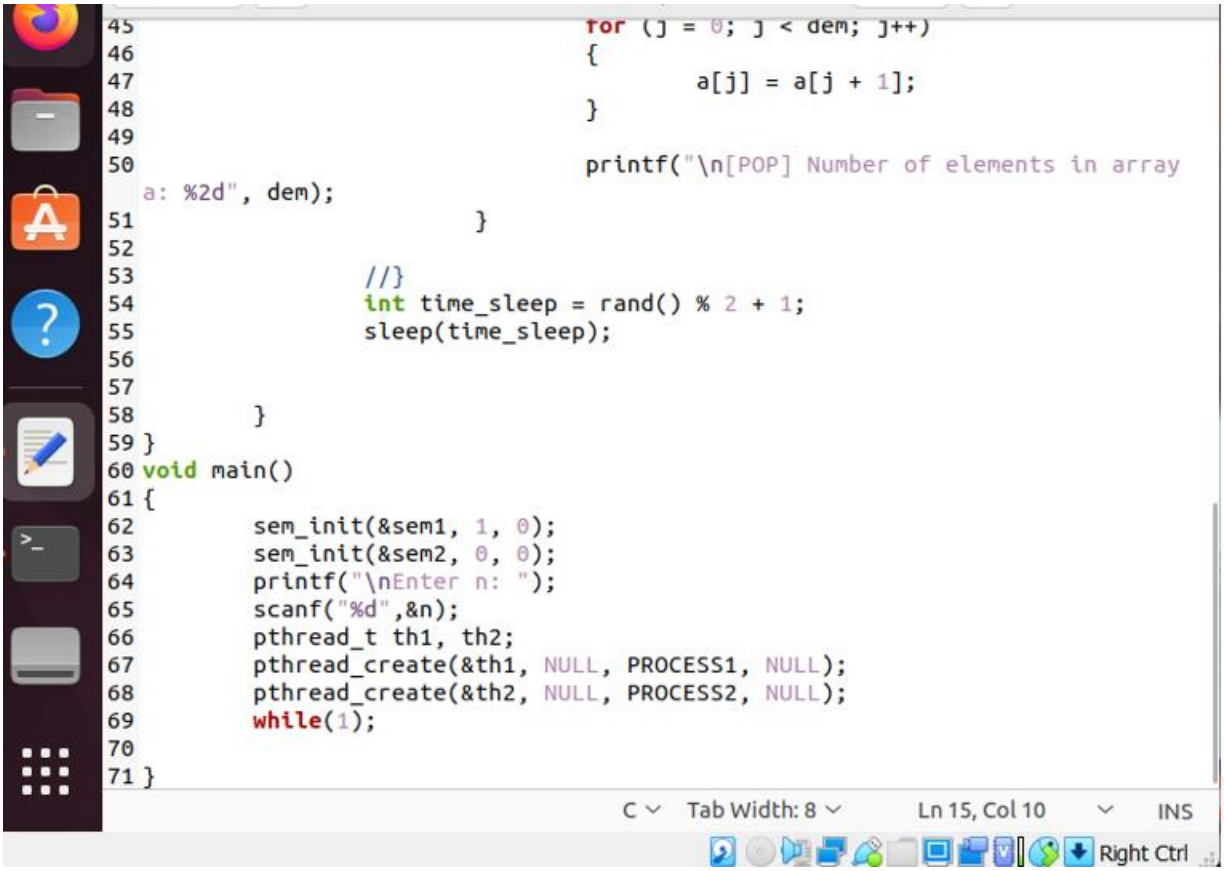




```
24                      sleep(time_sleep);
25
26                      sem_post(&sem1);
27              }
28 }
29 void* PROCESS2()
30 {
31              int j, b;
32              while (1)
33              {
34                      sem_wait(&sem1);
35                      //if (dem <= n) {
36
37                              if (dem == 0)
38                              {
39                                      printf("\n[POP] Nothing in array a");
40                              }
41                              else
42                              {
43                                      dem--;
44                                      b = a[0];
45                                      for (j = 0; j < dem; j++)
46                                      {
47                                              a[j] = a[j + 1];
48                                      }
49
50                                      printf("\n[POP] Number of elements in array
   a: %2d", dem);
```

C ▾ Tab Width: 8 ▾ Ln 15, Col 10 ▾ INS

Right Ctrl



```
45                                      for (j = 0; j < dem; j++)
46                                      {
47                                              a[j] = a[j + 1];
48                                      }
49
50                                      printf("\n[POP] Number of elements in array
   a: %2d", dem);
51                              }
52
53                      //}
54                      int time_sleep = rand() % 2 + 1;
55                      sleep(time_sleep);
56
57
58              }
59 }
60 void main()
61 {
62              sem_init(&sem1, 1, 0);
63              sem_init(&sem2, 0, 0);
64              printf("\nEnter n: ");
65              scanf("%d",&n);
66              pthread_t th1, th2;
67              pthread_create(&th1, NULL, PROCESS1, NULL);
68              pthread_create(&th2, NULL, PROCESS2, NULL);
69              while(1);
70
71 }
```

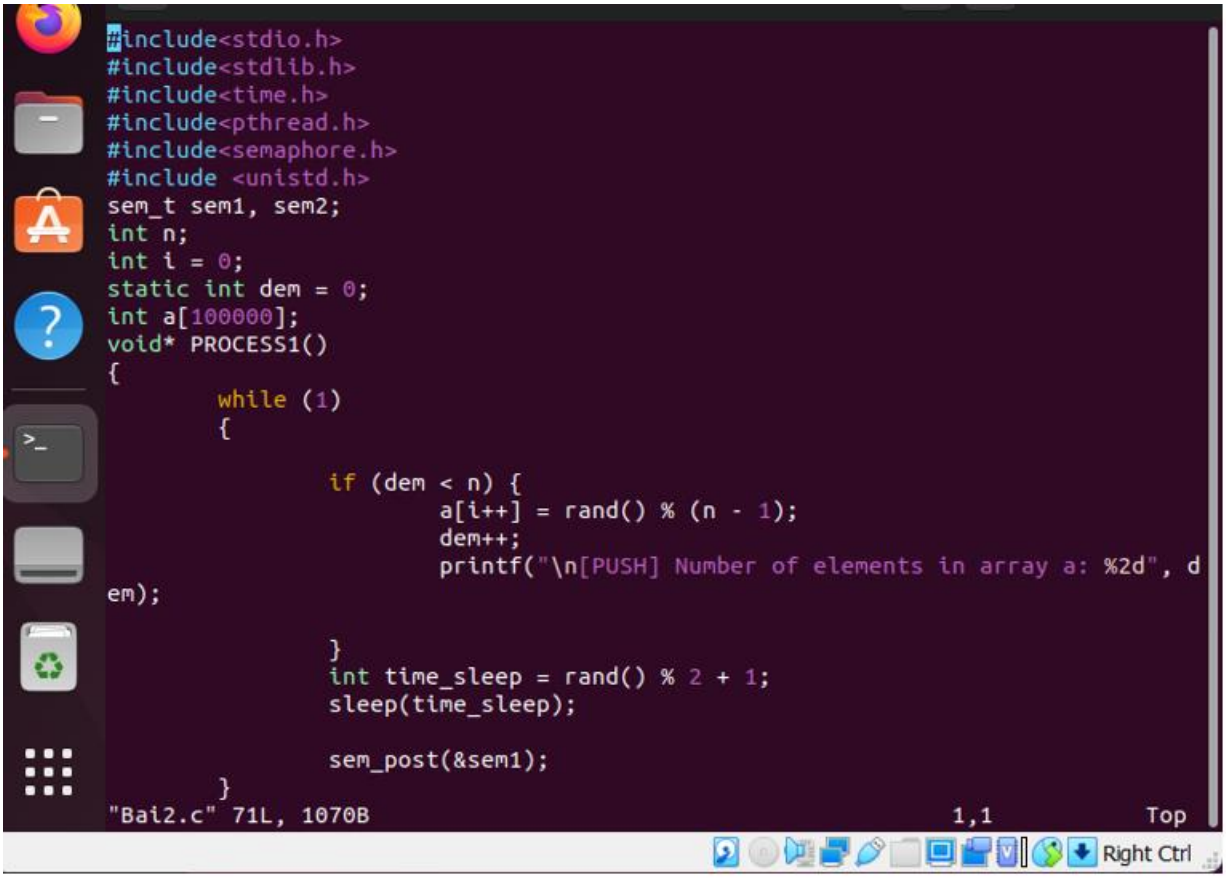
C ▾ Tab Width: 8 ▾ Ln 15, Col 10 ▾ INS

Right Ctrl

- **Bước 3:** Kiểm tra file Bai2.c đã được tạo thành công hay chưa (sử dụng câu lệnh $ ls)
    - ➢ Nếu chưa có file Bai2.c thì quay lại bước 1
    - ➢ Nếu đã có file Bai2.c thì tiếp tục đến bước 4

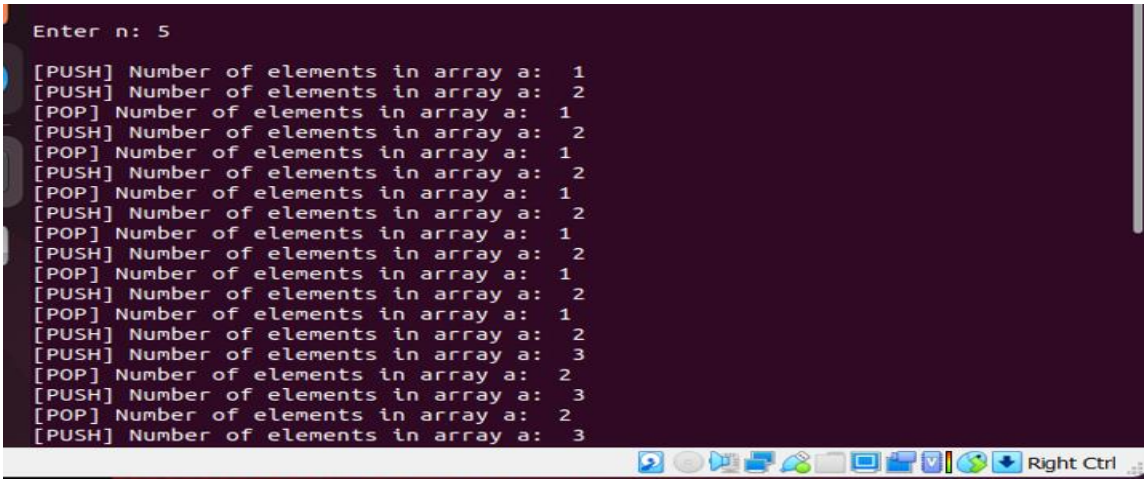
  File Bai2.c đã có trong danh sách => Tiếp tục đến bước 4

- **Bước 4:** Mở file Bai2.c bằng vim (sử dụng câu lệnh $ vim Bai2.c)



- **Bước 5:** Thực thi file Bai2.c (sử dụng câu lệnh $ gcc -pthread Bai2.c -o Bai2)

- **Bước 6:** Thực thi file Bai1 (sử dụng câu lệnh $ ./ Bai2)



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

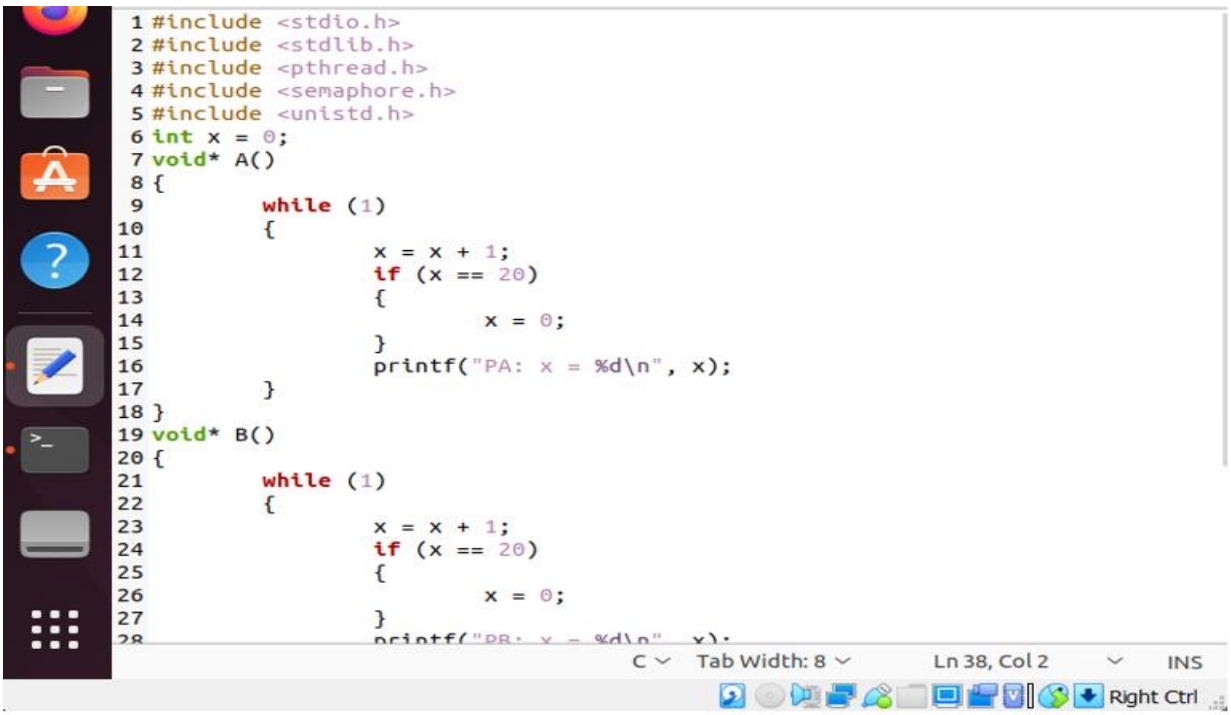
int x = 0;	
PROCESS A	**PROCESS B**
processA() {	processB() {

while(1){
x = x + 1;
if (x == 20)
x = 0;
print(x);
}
}
	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.

- **Bước 1:** Tạo file Bai3.c (sử dụng câu lệnh $ gedit Bai3.c)

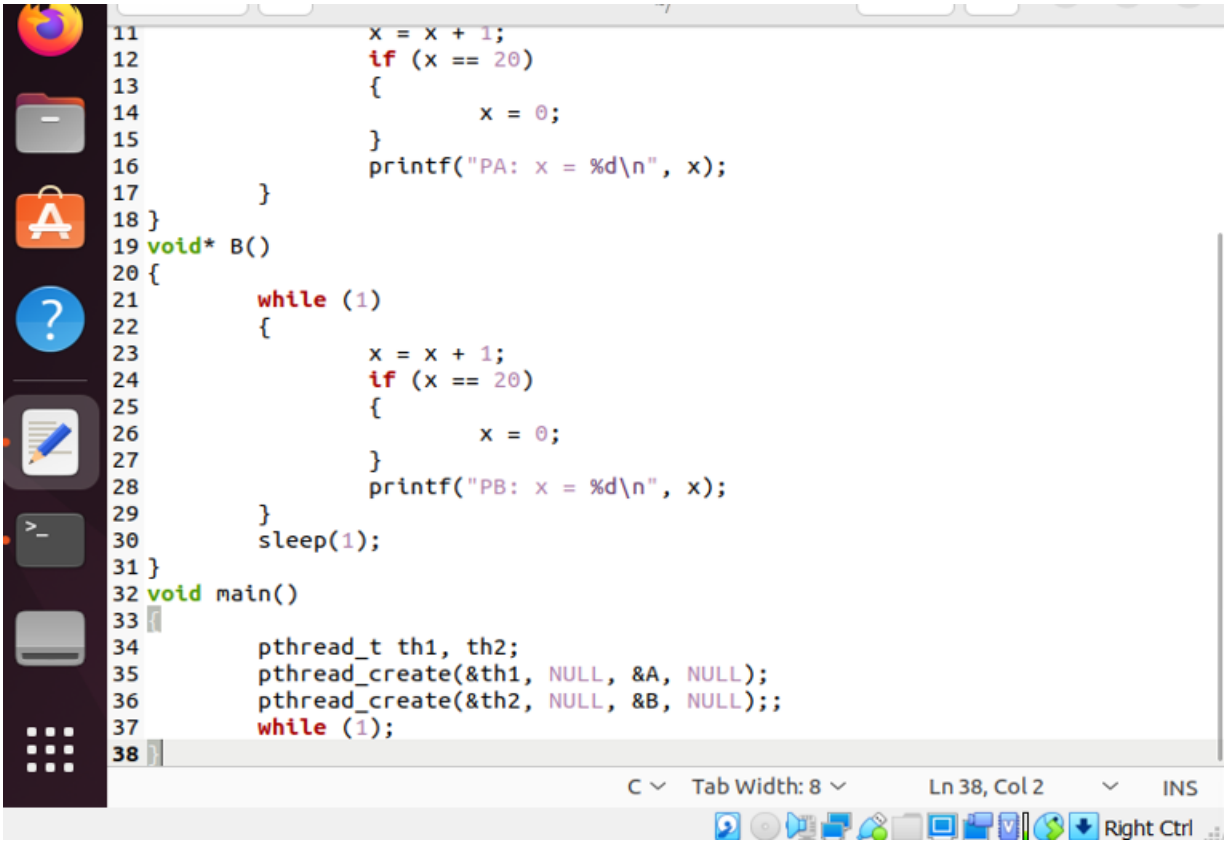
- **Bước 2:** Nhập nội dung cho file Bai3.c sau đó lưu lại



```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;
                }
```

C Tab Width: 8 Ln 38, Col 2 INS

Right Ctrl



```c
                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);
}
```

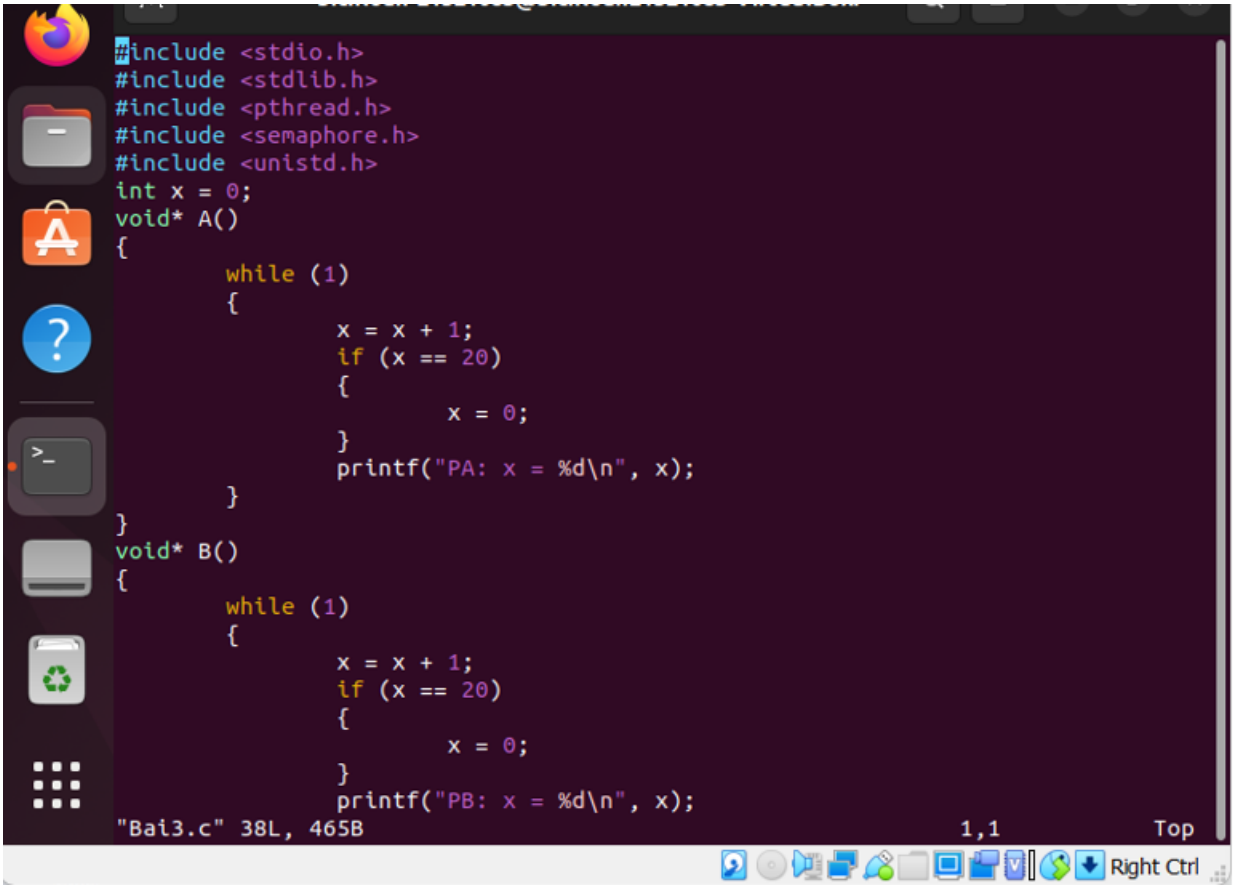
C Tab Width: 8 Ln 38, Col 2 INS

Right Ctrl

- **Bước 3:** Kiểm tra file Bai3.c đã được tạo thành công hay chưa (sử dụng câu lệnh $ ls)
    - ➢ Nếu chưa có file Bai3.c thì quay lại bước 1
    - ➢ Nếu đã có file Bai3.c thì tiếp tục đến bước 4

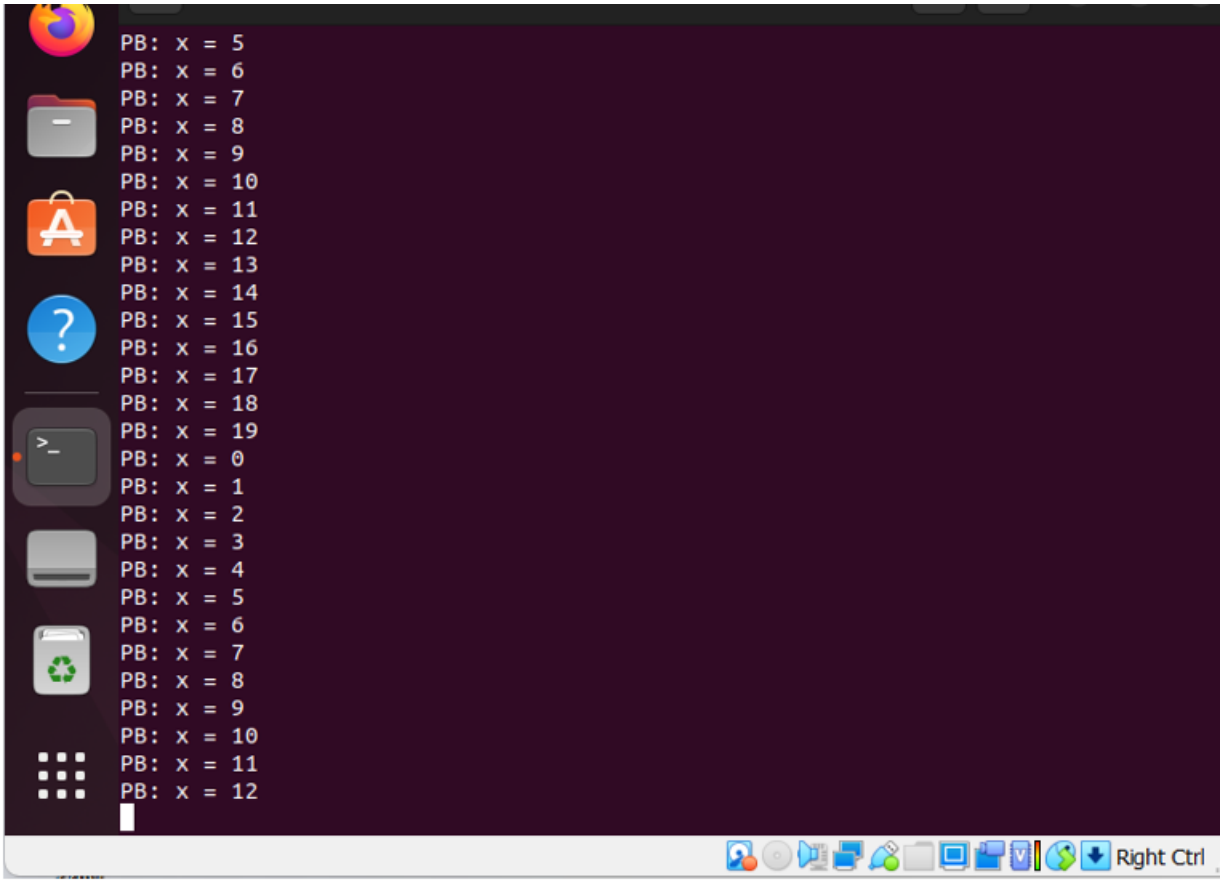
  File Bai3.c đã có trong danh sách => Tiếp tục đến bước 4

- **Bước 4:** Mở file Bai3.c bằng vim (sử dụng câu lệnh $ vim Bai3.c)



- **Bước 5:** Thực thi file Bai3.c (sử dụng câu lệnh $ gcc -pthread Bai3.c -o Bai3)

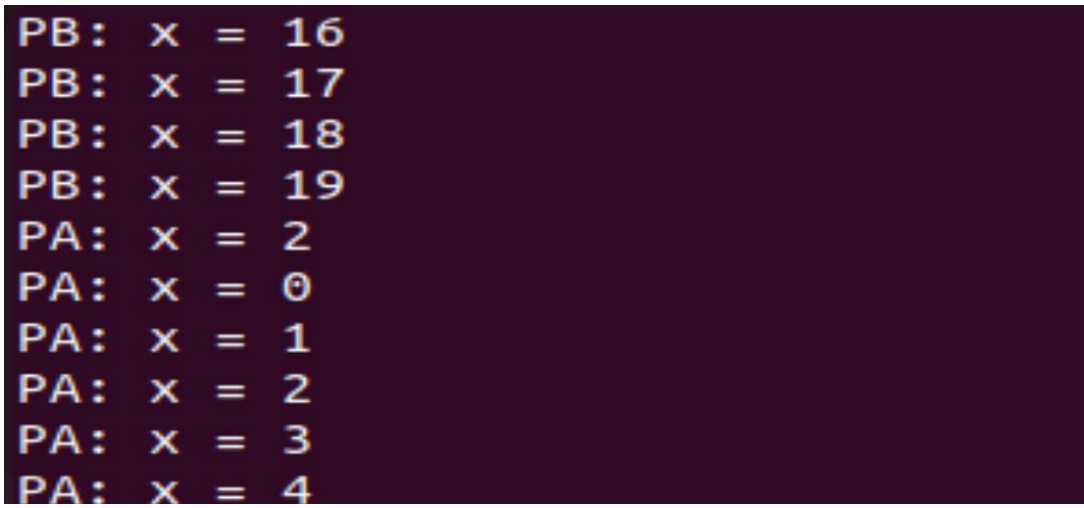
- **Bước 6:** Thực thi file Bai3 (sử dụng câu lệnh $ ./ Bai3)

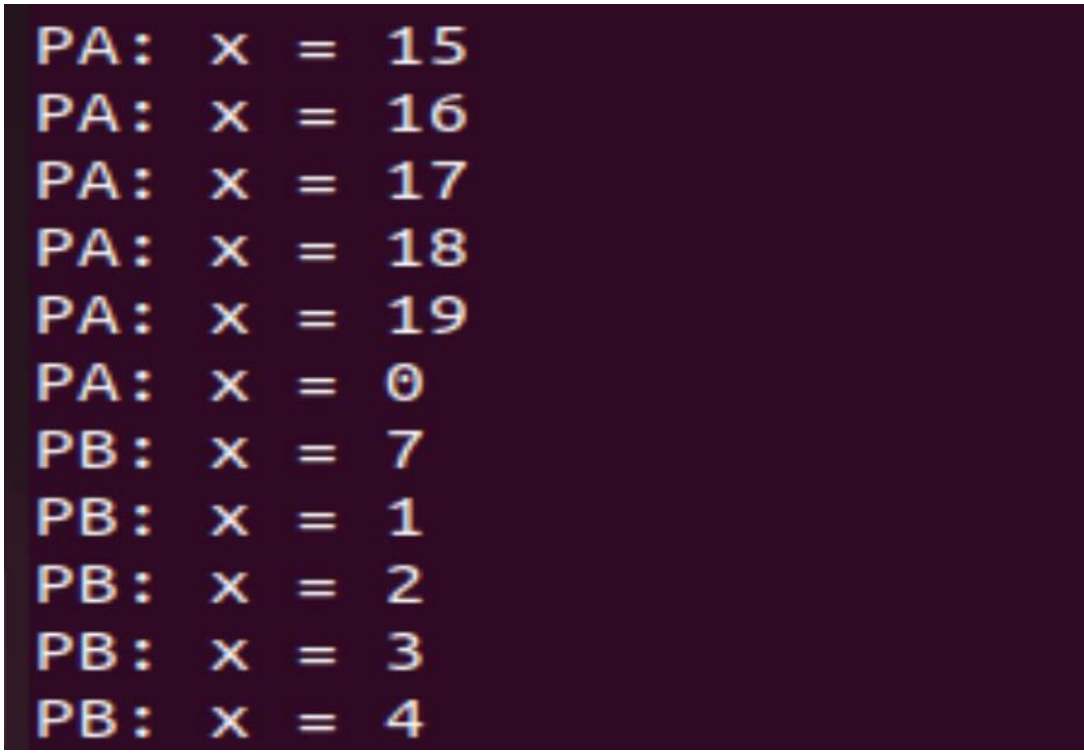


➢ Nhận xét:

+ Trong quá trình chạy có trường hợp biến B đang chạy dở (ví dụ biến B chạy đến giá trị 6) sau đó nhường cho biến A chạy (thì biến A có giá trị là 1) nhưng sau đó biến A sẽ chạy từ giá trị hiện tại của biến B (là giá trị 7), tiếp tục chạy theo vòng lặp cho đến hết chu kỳ của A (giả sử biến A đang có giá trị là 10).

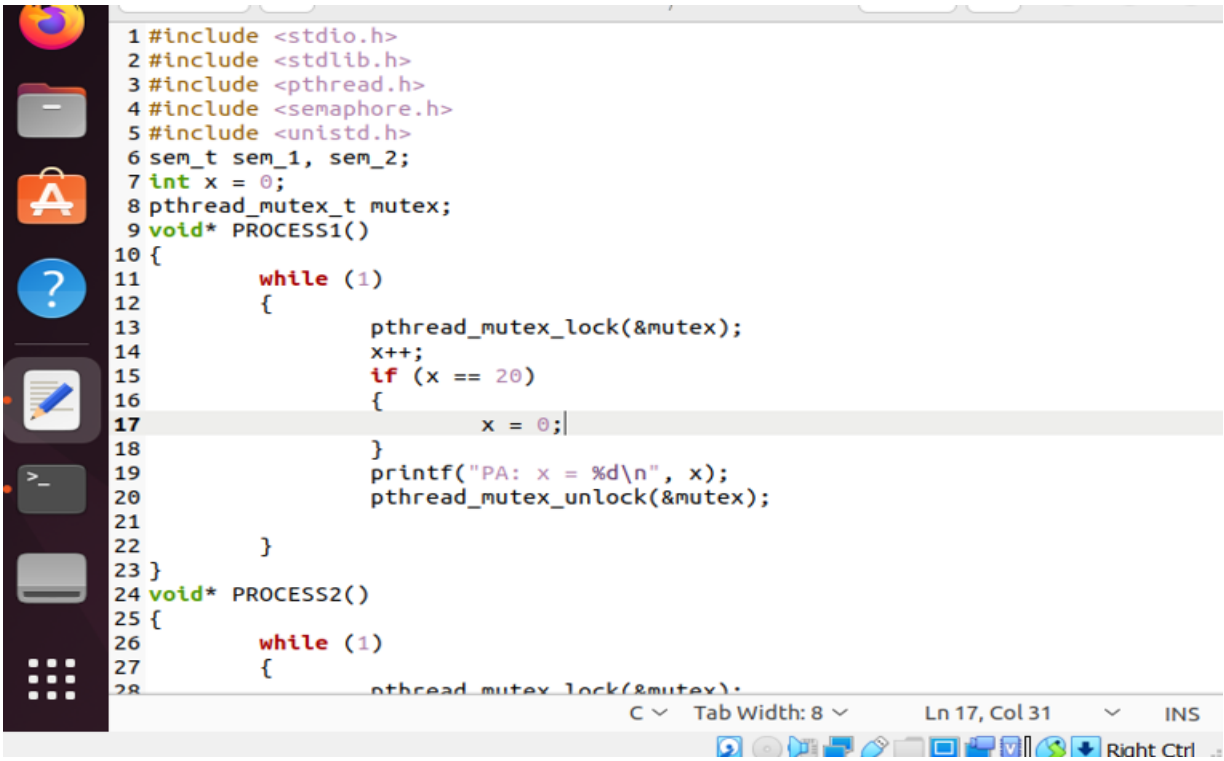
+ Biến B bắt đầu lại từ giá trị 7 sau đó lại chạy theo kết quả của biến A. Lúc này biến A có giá trị là 11





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

- **Bước 7:** Tạo file Bai4.c (sử dụng câu lệnh $ gedit Bai4.c)
- **Bước 8:** Nhập nội dung cho file Bai4.c sau đó lưu lại

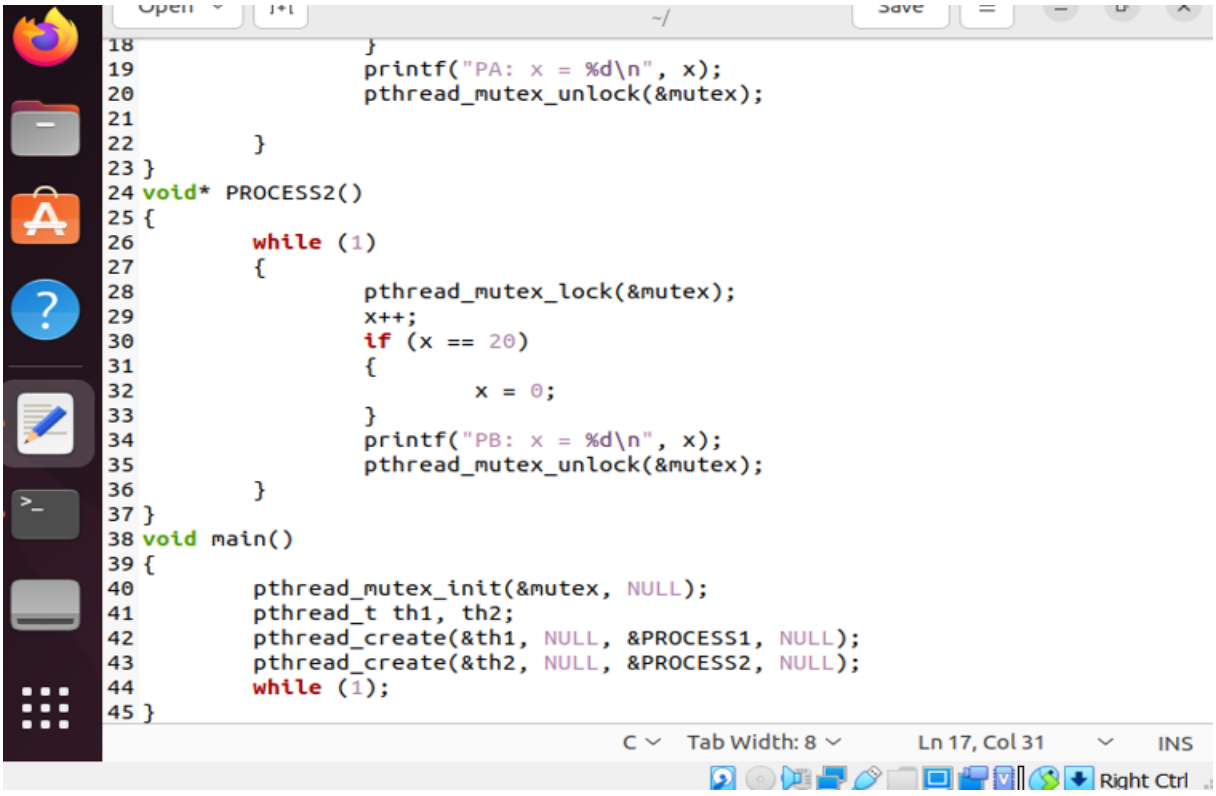


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t sem_1, sem_2;
int x = 0;
pthread_mutex_t mutex;
void* PROCESS1()
{
        while (1)
        {
                pthread_mutex_lock(&mutex);
                x++;
                if (x == 20)
                {
                        x = 0;
                }
                printf("PA: x = %d\n", x);
                pthread_mutex_unlock(&mutex);

        }
}
void* PROCESS2()
{
        while (1)
        {
```

C Tab Width: 8 Ln 17, Col 31 INS

Right Ctrl



```c
                }
                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);
}
```

C Tab Width: 8 Ln 17, Col 31 INS

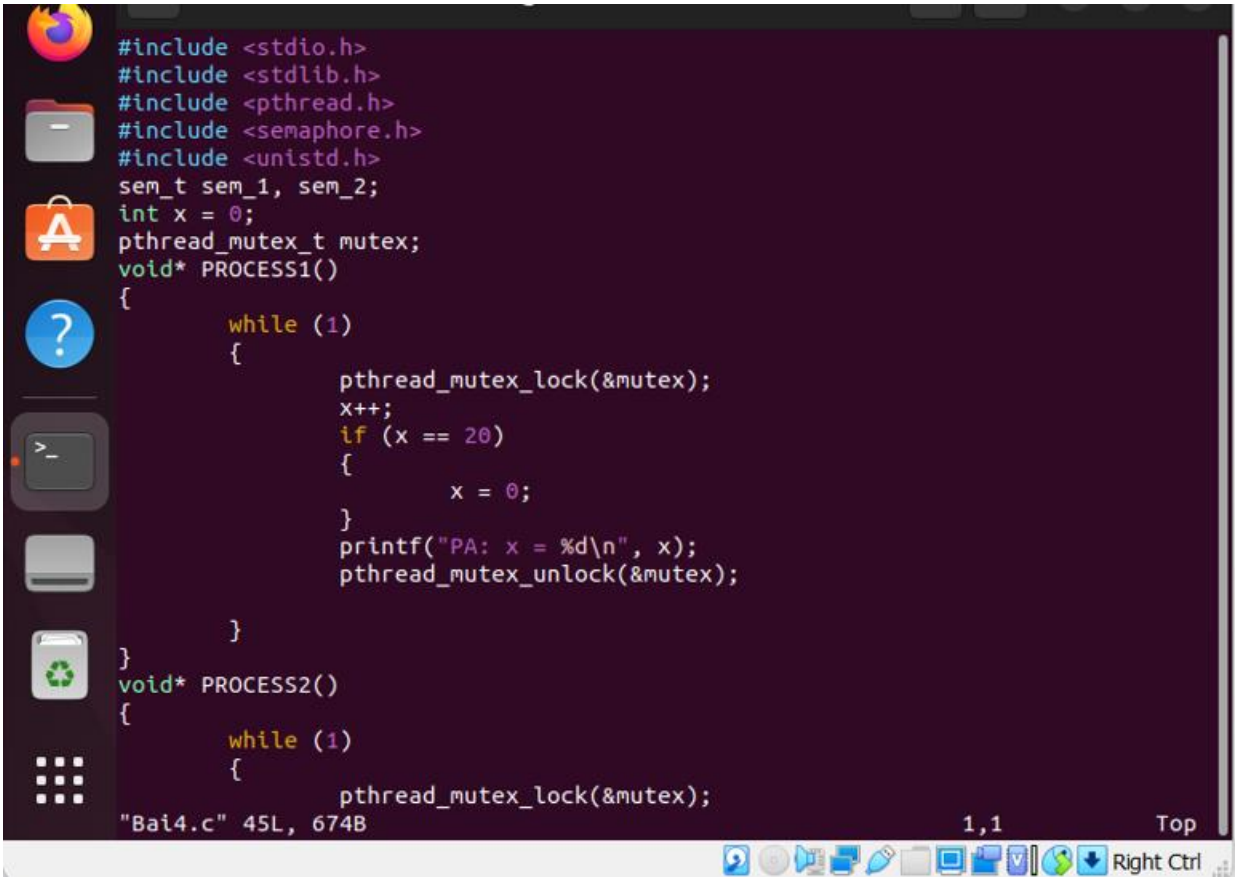
Right Ctrl

- **Bước 9:** Kiểm tra file Bai4.c đã được tạo thành công hay chưa (sử dụng câu lệnh $ ls)

  - Nếu chưa có file Bai4.c thì quay lại bước 1
  - Nếu đã có file Bai4.c thì tiếp tục đến bước 4

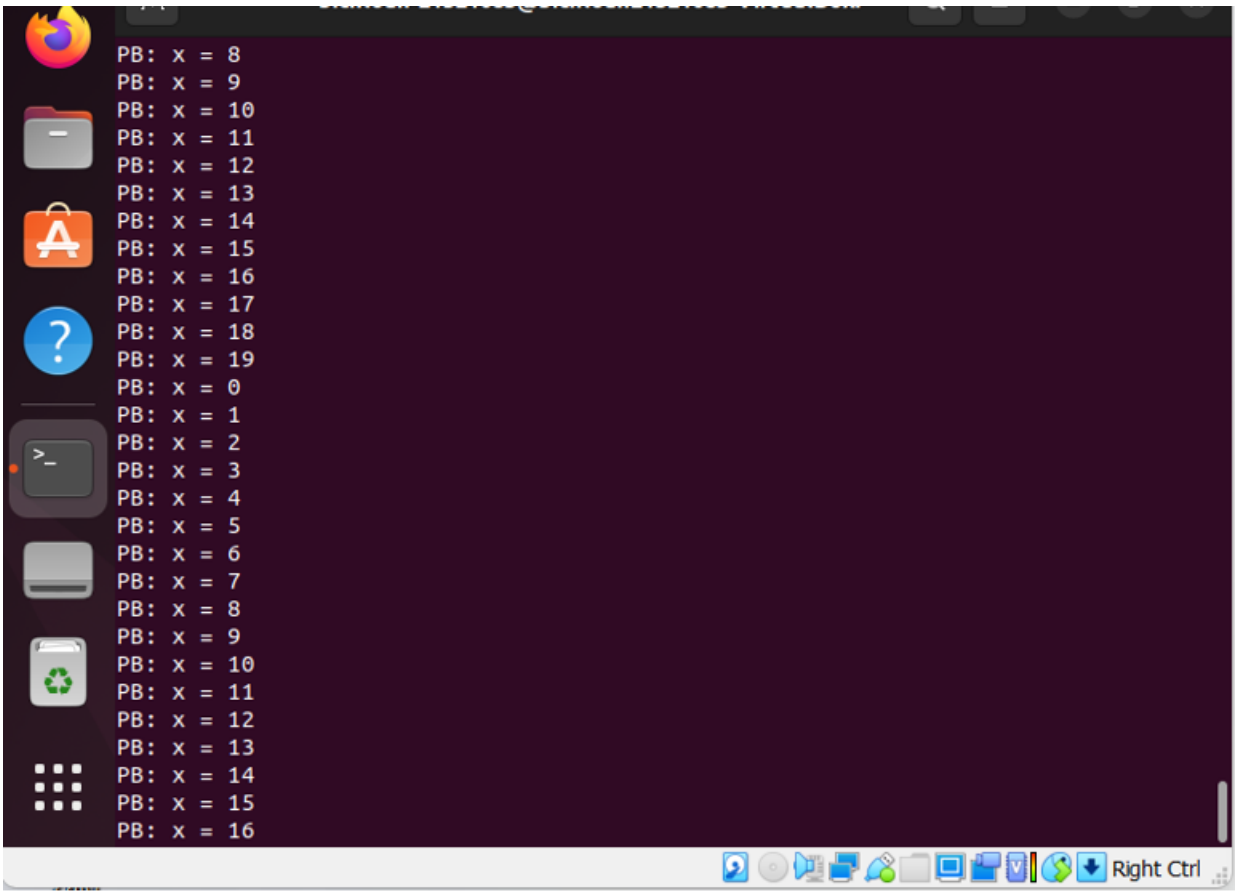
  File Bai4.c đã có trong danh sách => Tiếp tục đến bước 4

- **Bước 10:** Mở file Bai4.c bằng vim (sử dụng câu lệnh $ vim Bai4.c)



- **Bước 11:** Thực thi file Bai4.c (sử dụng câu lệnh $ gcc -pthread Bai4.c -o Bai4)

- **Bước 12:** Thực thi file Bai1 (sử dụng câu lệnh $ ./ Bai4)


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 = 9
PB: x = 10
PB: x = 11
PB: x = 12
PB: x = 13
PB: x = 14
PB: x = 15
PB: x = 16
Right Ctrl

5.5 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)

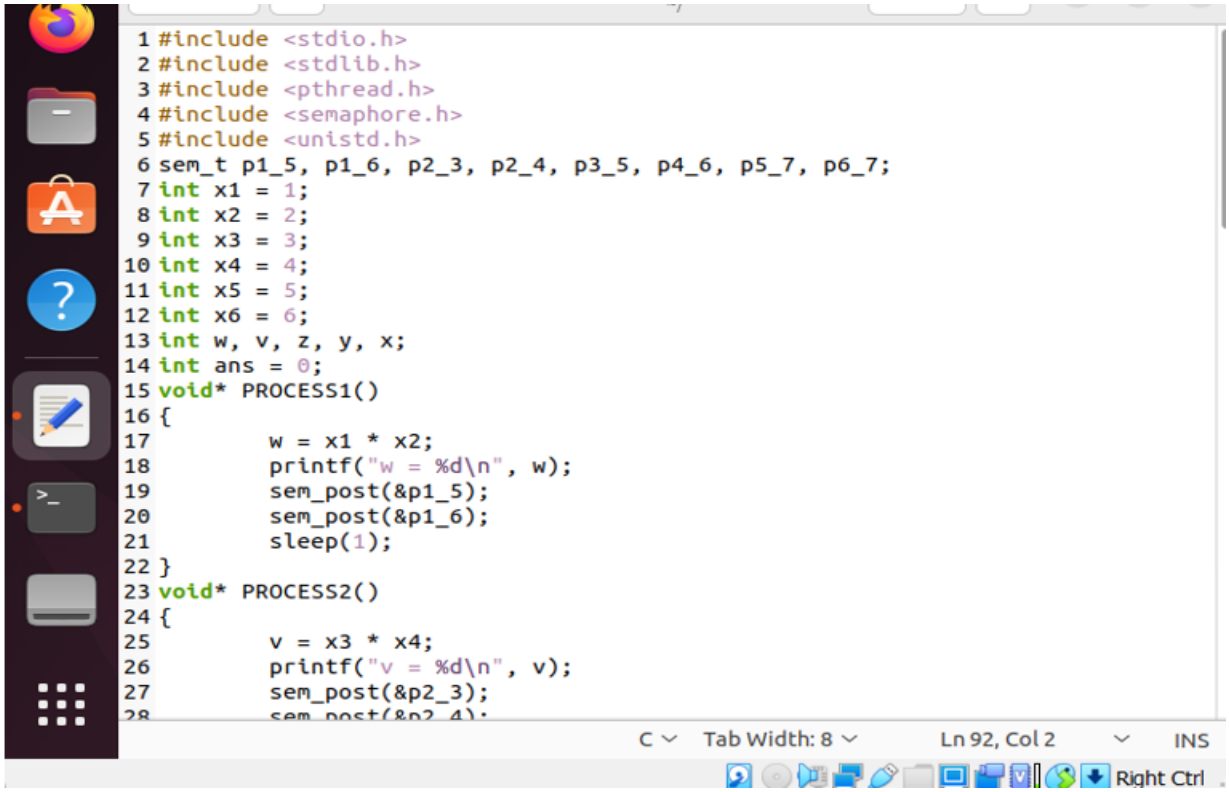
z = w * z; (f)

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

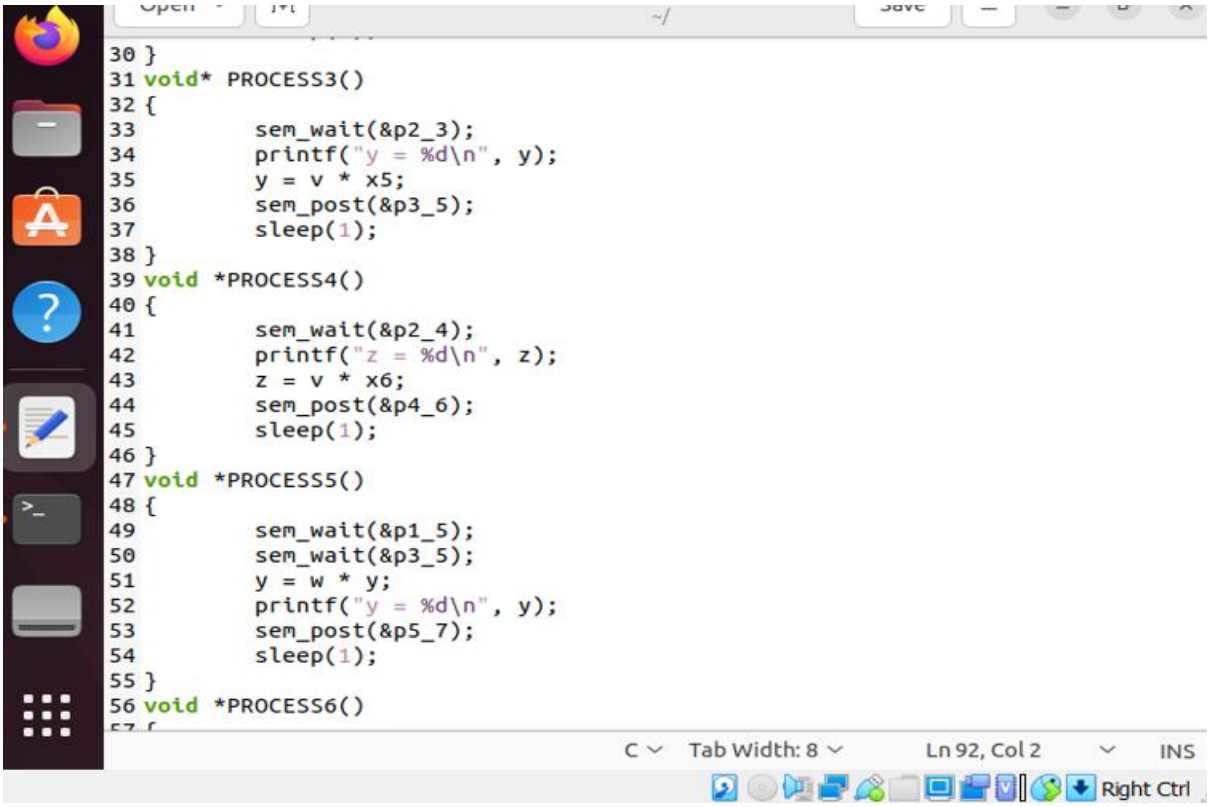
- **Bước 13:** Tạo file Bai5.c (sử dụng câu lệnh $ gedit Bai5.c)
- **Bước 14:** Nhập nội dung cho file Bai5.c sau đó lưu lại



```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);
```

C Tab Width: 8 Ln 92, Col 2 INS

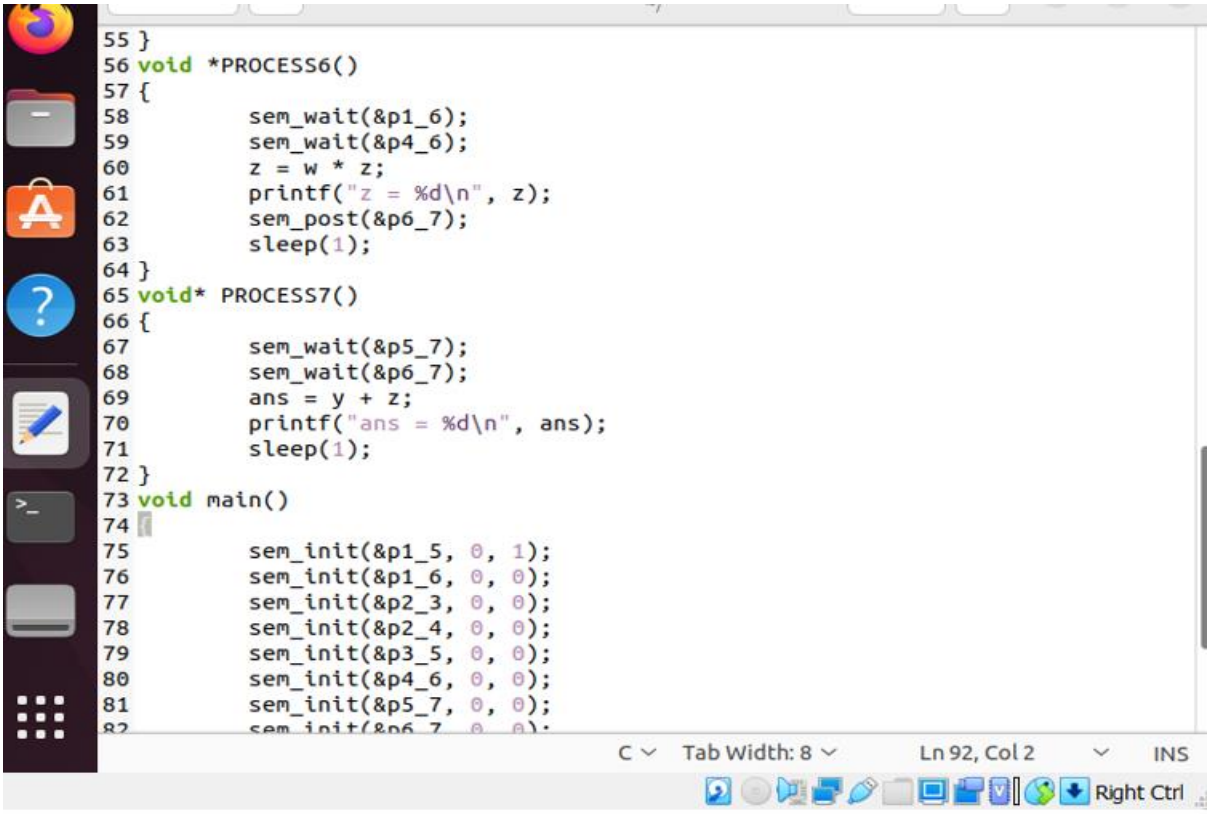
Right Ctrl



```c
}
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()
```

C Tab Width: 8 Ln 92, Col 2 INS

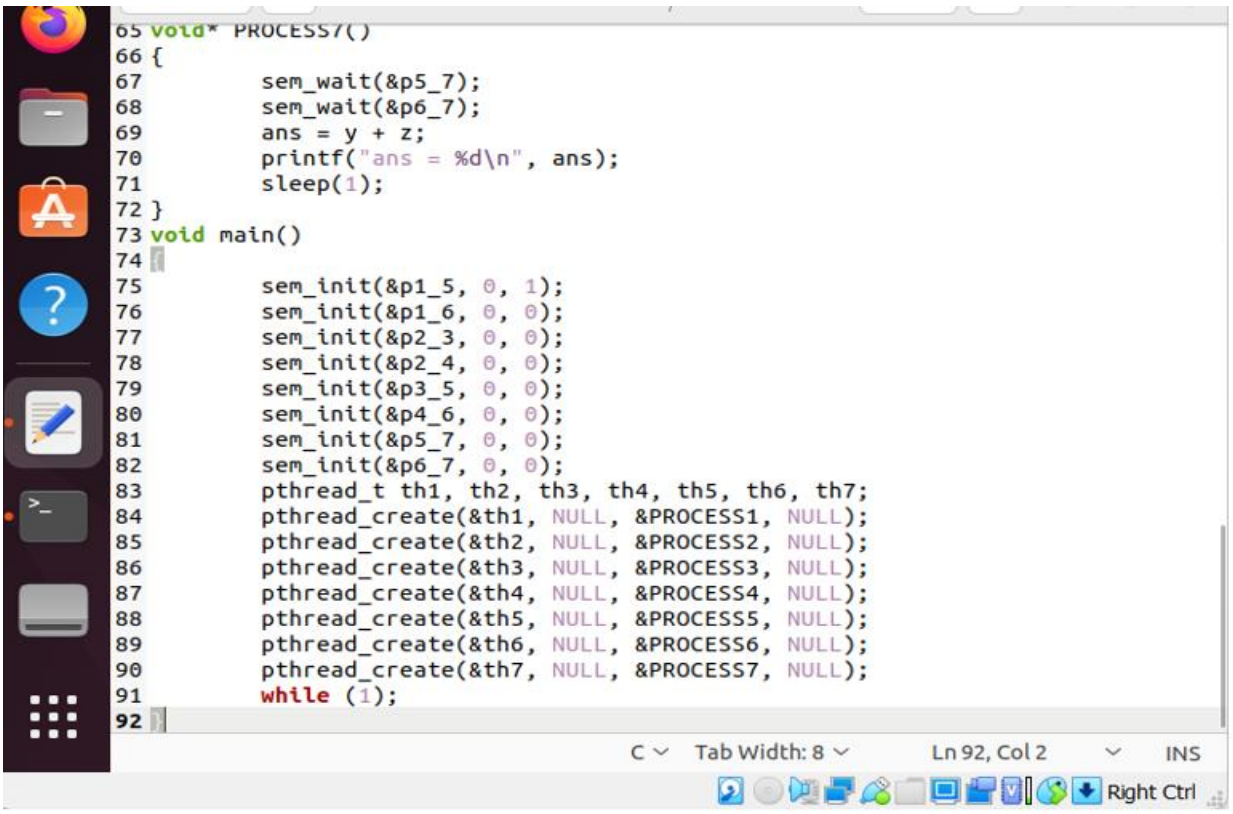
Right Ctrl



```
55 }
56 void *PROCESS6()
57 {
58         sem_wait(&p1_6);
59         sem_wait(&p4_6);
60         z = w * z;
61         printf("z = %d\n", z);
62         sem_post(&p6_7);
63         sleep(1);
64 }
65 void* PROCESS7()
66 {
67         sem_wait(&p5_7);
68         sem_wait(&p6_7);
69         ans = y + z;
70         printf("ans = %d\n", ans);
71         sleep(1);
72 }
73 void main()
74 {
75         sem_init(&p1_5, 0, 1);
76         sem_init(&p1_6, 0, 0);
77         sem_init(&p2_3, 0, 0);
78         sem_init(&p2_4, 0, 0);
79         sem_init(&p3_5, 0, 0);
80         sem_init(&p4_6, 0, 0);
81         sem_init(&p5_7, 0, 0);
```

C Tab Width: 8 Ln 92, Col 2 INS

Right Ctrl



```
65 void* PROCESS7()
66 {
67         sem_wait(&p5_7);
68         sem_wait(&p6_7);
69         ans = y + z;
70         printf("ans = %d\n", ans);
71         sleep(1);
72 }
73 void main()
74 {
75         sem_init(&p1_5, 0, 1);
76         sem_init(&p1_6, 0, 0);
77         sem_init(&p2_3, 0, 0);
78         sem_init(&p2_4, 0, 0);
79         sem_init(&p3_5, 0, 0);
80         sem_init(&p4_6, 0, 0);
81         sem_init(&p5_7, 0, 0);
82         sem_init(&p6_7, 0, 0);
83         pthread_t th1, th2, th3, th4, th5, th6, th7;
84         pthread_create(&th1, NULL, &PROCESS1, NULL);
85         pthread_create(&th2, NULL, &PROCESS2, NULL);
86         pthread_create(&th3, NULL, &PROCESS3, NULL);
87         pthread_create(&th4, NULL, &PROCESS4, NULL);
88         pthread_create(&th5, NULL, &PROCESS5, NULL);
89         pthread_create(&th6, NULL, &PROCESS6, NULL);
90         pthread_create(&th7, NULL, &PROCESS7, NULL);
91         while (1);
92 }
```

C Tab Width: 8 Ln 92, Col 2 INS

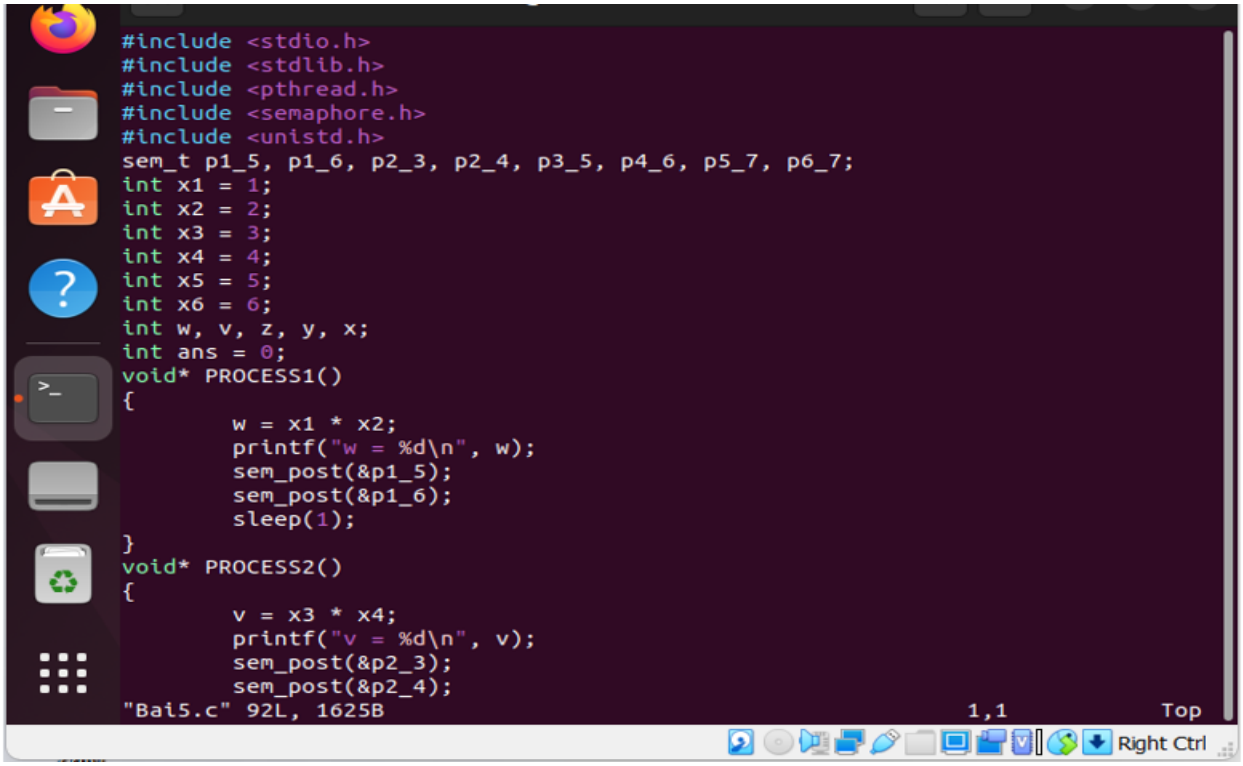
Right Ctrl

- **Bước 15:** Kiểm tra file Bai5.c đã được tạo thành công hay chưa (sử dụng câu lệnh $ ls)

  - Nếu chưa có file Bai5.c thì quay lại bước 1
  - Nếu đã có file Bai5.c thì tiếp tục đến bước 4

  File Bai5.c đã có trong danh sách => Tiếp tục đến bước 4

- **Bước 16:** Mở file Bai5.c bằng vim (sử dụng câu lệnh $ vim Bai5.c)



- **Bước 17:** Thực thi file Bai5.c (sử dụng câu lệnh $ gcc -pthread Bai5.c -o Bai5)

- **Bước 18:** Thực thi file Bai5 (sử dụng câu lệnh $ ./ Bai5)

5.6 Câu hỏi thêm

1. So sánh tiến trình và tiểu trình. Khi nào sử dụng tiến trình, khi nào sử dụng tiểu trình. Ưu, nhược điểm

	Tiến trình	Tiểu trình
Ưu điểm	+ Tiến trình là một bộ phận của chương trình đang thực hiện. + Tiến trình là đơn vị làm việc cơ bản của hệ thống, trong hệ thống có thể tồn tại nhiều tiến trình cùng hoạt động, trong đó có cả tiến trình của hệ điều hành và tiến trình của chương trình người sử dụng. + Các tiến trình có thể hoạt động đồng thời với nhau.	+ Tiểu trình cũng là đơn vị xử lý cơ bản trong hệ thống, nó cũng xử lý tuần tự đoạn code của nó, nó cũng 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. + Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung, điều này có nghĩa các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình, có thể truy xuất đến stack của tiểu trình khác trong cùng tiến trình.
Nhược điểm	+ Để một tiến trình đi vào trạng thái hoạt động thì hệ thống phải cung cấp đầy đủ tài nguyên cho tiến trình. + Hệ thống cũng phải duy trì đủ tài nguyên cho tiến trình trong suốt quá trình hoạt động của tiến trình.	+ Với mô hình tiểu trình, trong hệ thống có thể tồn tại nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ bộ nhớ, các dòng xử lý này hoạt động song song với nhau.

		+ Nếu tiến trình không hoạt động thì tiểu trình cũng bị buộc dừng

2. So sánh mutex và semaphore

Semaphore	Mutex
+ Là 1 biến được sử dụng để điều khiển sự truy xuất vào các tài nguyên chung của 2 tiểu trình trong xử lý song song hoặc các môi trường đa người dùng. + Semaphore được xem như một danh sách các đơn vị còn trống của một tài nguyên trong máy tính. + Có 2 thao tác cơ bản trên semaphore là yêu cầu tài nguyên và giải phóng tài nguyên. + Nếu cần thiết, semaphore còn có thể làm cờ để đợi cho đến khi tài nguyên được một tiểu trình khác giải phóng. + Semaphore là một cơ chế báo hiệu. + Semaphore là một biến số nguyên. + Semaphore cho phép nhiều luồng chương trình truy cập vào một thể hiện hữu hạn của tài nguyên. + Giá trị semaphore có thể được thay đổi bởi bất kỳ quá trình có được hoặc giải phóng tài nguyên.	+ Mutex là một trường hợp đơn giản của semaphore: 0 <= sem.value <= 1 + Mutex là một cơ chế khóa. + Mutex là một đối tượng. + Mutex cho phép nhiều luồng chương trình truy cập vào một tài nguyên nhưng không đồng thời. + Khóa đối tượng Mutex chỉ được phát hành bởi quá trình đã có được khóa trên nó. + Mutex không được phân loại thêm. + Đối tượng Mutex bị khóa hoặc mở khóa bởi quá trình yêu cầu hoặc giải phóng tài nguyên. + Nếu một đối tượng mutex đã bị khóa, quá trình yêu cầu tài nguyên chờ và được hệ thống xếp hàng cho đến khi khóa được giải phóng.

+ Semaphore có thể được phân loại thành đếm semaphore và semaphore nhị phân. + Giá trị semaphore được sửa đổi bằng cách sử dụng thao tác Wait () và signal (). + Nếu tất cả các tài nguyên đang được sử dụng, quá trình yêu cầu tài nguyên thực hiện thao tác Wait () và tự chặn cho đến khi số lượng semaphore trở nên lớn hơn một.	