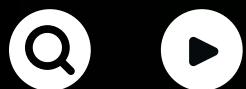




SISTEMAS OPERATIVOS

MECANISMOS PARA MANTENER LA COHERENCIA EN CACHÉ



Elizabeth Portilla

Maria Fernanda Ayala

Slide 1



Antes, la velocidad del procesador estaba sincronizada con la del manejador de memoria, y el flujo podía mantenerse estable.

Con el tiempo la CPU se volvió mucho más rápida que la memoria, generando una brecha de rendimiento.

El CPU no puede, a diferencia del sistema operativo, "congelar" todo y guardar el estado para atender otro proceso, para el procesador, la lista de instrucciones a ejecutar es estrictamente secuencial



Memoria caché

Memoria temporal, de menor capacidad y de rápido acceso.

Almacena una copia de los datos de uso frecuente.

Primero se busca en la memoria caché.

Fundamental que el contenido de la memoria caché y el de la memoria principal se mantengan consistentes.

Mínima unidad de intercambio de datos entre la memoria caché y la principal es la línea de caché.

Situada entre la memoria principal y el procesador

Si los datos están allí (cache hit).

Si no están (cache miss).

Cualquier desincronización puede afectar negativamente el rendimiento del sistema.





Coherencia en caché

Se refiere a la consistencia y sincronización de los datos almacenados en diferentes memorias caché dentro de un sistema multiprocesador o multinúcleo.

En sistemas multiprocesador o multinúcleo, cada procesador tiene su propia caché. Este mecanismo mejora notablemente el rendimiento, pero también introduce un desafío: mantener la consistencia de los datos compartidos.



Si varias cachés almacenan copias del mismo bloque de memoria y una de ellas lo modifica, deben actualizarse las demás para evitar inconsistencias. Este fenómeno se conoce como el problema de coherencia de caché. Si no se controla, puede generar errores en la ejecución del software o incluso corrupción de datos.



Mecanismos de coherencia en caché

¿Qué son?

Son protocolos que se usan en sistemas con varios procesadores o núcleos para asegurarse de que todos vean los mismos datos, aunque cada uno tenga su propia caché.

Ayudan a que las cachés se mantengan sincronizadas y evitan errores cuando varios núcleos comparten y modifican la misma información.



Fisgones (snooping)



Por directorio



Otros protocolos basados en snooping:

- **MSI**
- **MESI**
- **MOSI**
- **MOESI**
- **MESIF**

Snooping

Son protocolos que permiten mantener la coherencia entre cachés en sistemas multiprocesador, usando un bus compartido.

¿Cómo funciona?

- ◆ Snooping significa que cada caché está “espiando” el bus para ver si otro procesador hace cambios en los datos que también tiene guardados.
- ◆ Si detecta que otro procesador modificó un dato que también tiene, marca su copia como inválida. A esto se le llama invalidación cruzada.
- ◆ Esa invalidez indica que ya no puede confiar en su copia y debe buscar el dato actualizado en otra parte (en otra caché o en memoria principal).



Problemas de Rendimiento

- Esta técnica genera mucho tráfico en el bus porque todas las cachés están monitoreando constantemente.
- Las invalidaciones cruzadas hacen que las cachés fallen más seguido, porque pierden datos válidos y deben recargarlos.
- Todo esto puede reducir el rendimiento general del sistema si no se gestiona bien.

Políticas

Write-Invalidate

Esta política se basa en invalidar las copias de un bloque en otras cachés cuando un procesador quiere escribir en él.

Antes de escribir, se manda una señal para que todas las demás copias se marquen como inválidas. Así, solo una caché tiene permiso para modificar ese bloque.

Write-Update

En lugar de invalidar las copias, esta política actualiza todas las copias del bloque en las demás cachés cada vez que se escribe.

- Requiere más ancho de banda, ya que se transmite el nuevo valor a todas las cachés.



Directory-based

Son una forma de mantener los datos consistentes en sistemas multiprocesador cuando no se puede usar un bus compartido, como ocurre en sistemas con redes punto a punto o multietapa.

👉 En lugar del bus, estos sistemas usan un directorio que lleva un registro de qué procesadores tienen copias de cada bloque de memoria y en qué estado están.

¿Cómo funciona?

- Una caché quiere leer o escribir un bloque.
- Consulta al directorio si puede hacerlo.
- El directorio coordina el acceso y notifica a las demás cachés si deben invalidar o actualizar sus copias.
- Esto evita inconsistencias entre procesadores.

Tipos de Organización del Directorio

• Directorio centralizado

1.Toda la información está en un solo lugar.

2.Puede generar un cuello de botella si muchos procesadores lo consultan al mismo tiempo.

• Directorio distribuido

1.Cada módulo de memoria tiene su propio directorio.

2.Mejor distribución de la carga y escalabilidad.

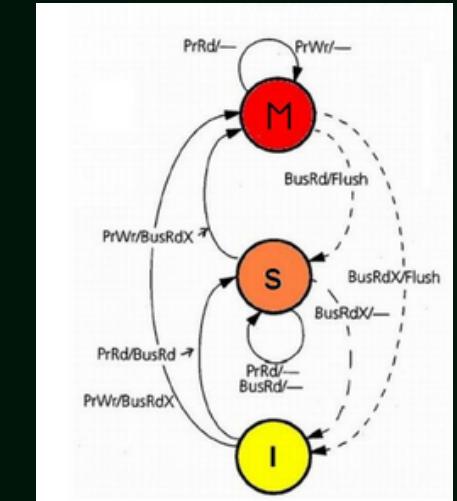


Otros protocolos

MSI

MSI (Modified – Shared – Invalid)

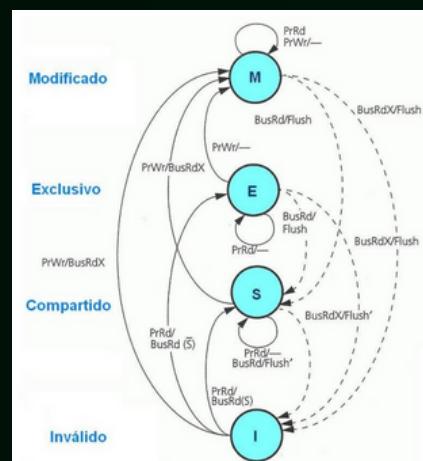
- M (Modified): El bloque fue cambiado en la caché. Solo está en una caché y no coincide con la memoria.
- S (Shared): Hay copias en varias cachés y coincide con la memoria. Solo se permite lectura.
- I (Invalid): El bloque no es válido, debe traerse de memoria o de otra caché.



MESI

MESI (agrega el estado Exclusive)

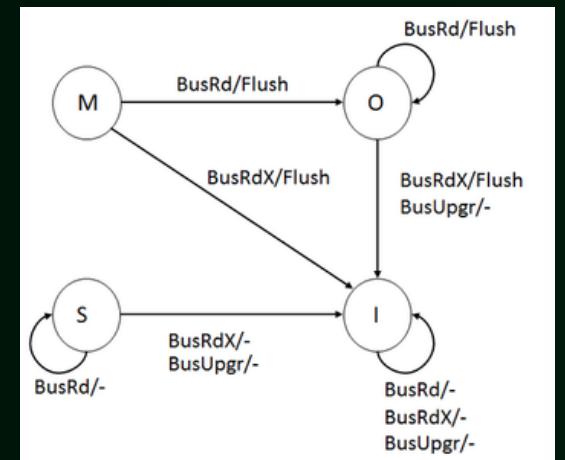
- E (Exclusive): El bloque está solo en una caché y coincide con la memoria.
- Si se modifica, pasa a Modified sin avisar a otras cachés, ahorrando tráfico.
- Ventaja: Menos tráfico en el bus al modificar bloques que solo tiene una caché.



MOSI

MOSI (agrega el estado Owned)

- O (Owned): El bloque fue modificado, pero se permite compartirlo.
 - Esta caché responde a otras que lo pidan, sin necesidad de escribirlo en memoria.
- Ventaja: Se evita escribir inmediatamente en memoria cuando se comparte un bloque modificado.

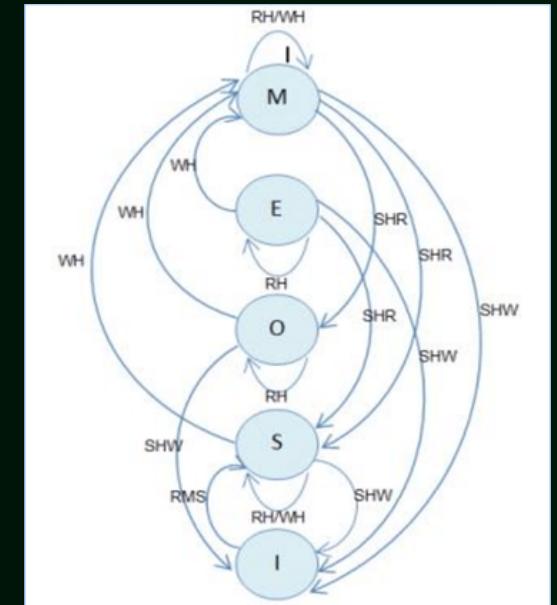




MOESI

MOESI (une MESI + Owned)

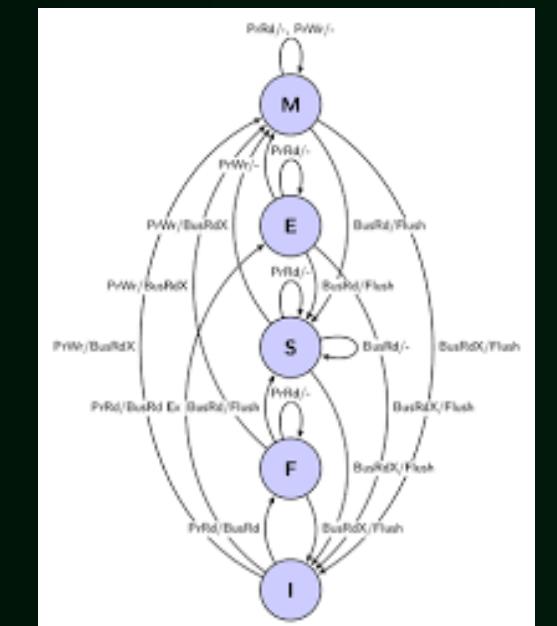
- Mezcla lo mejor de MESI y MOSI.
- Permite que una caché comparta un bloque modificado directamente a otra, sin pasar por la memoria.
- Mejor rendimiento: Menos tráfico, más reutilización de datos entre procesadores.



MESIF

MESIF (agrega el estado Forward)

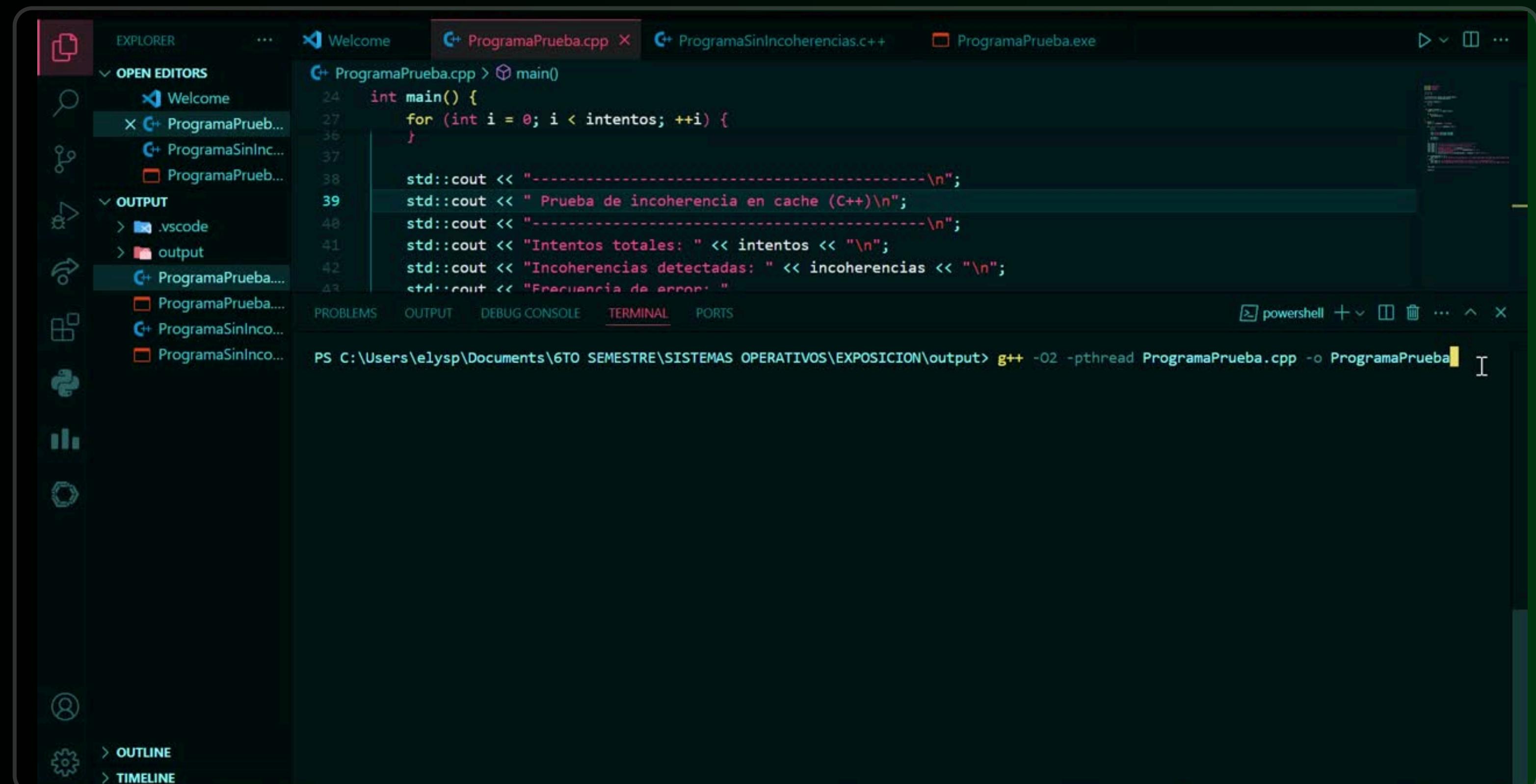
- F (Forward): Entre las cachés en estado Shared, una es la "reenviadora oficial".
- Solo ella responde a solicitudes externas del bloque.
- Ventaja: Se evita que varias cachés respondan al mismo tiempo, reduciendo conflictos y tráfico.





Actividad planeada





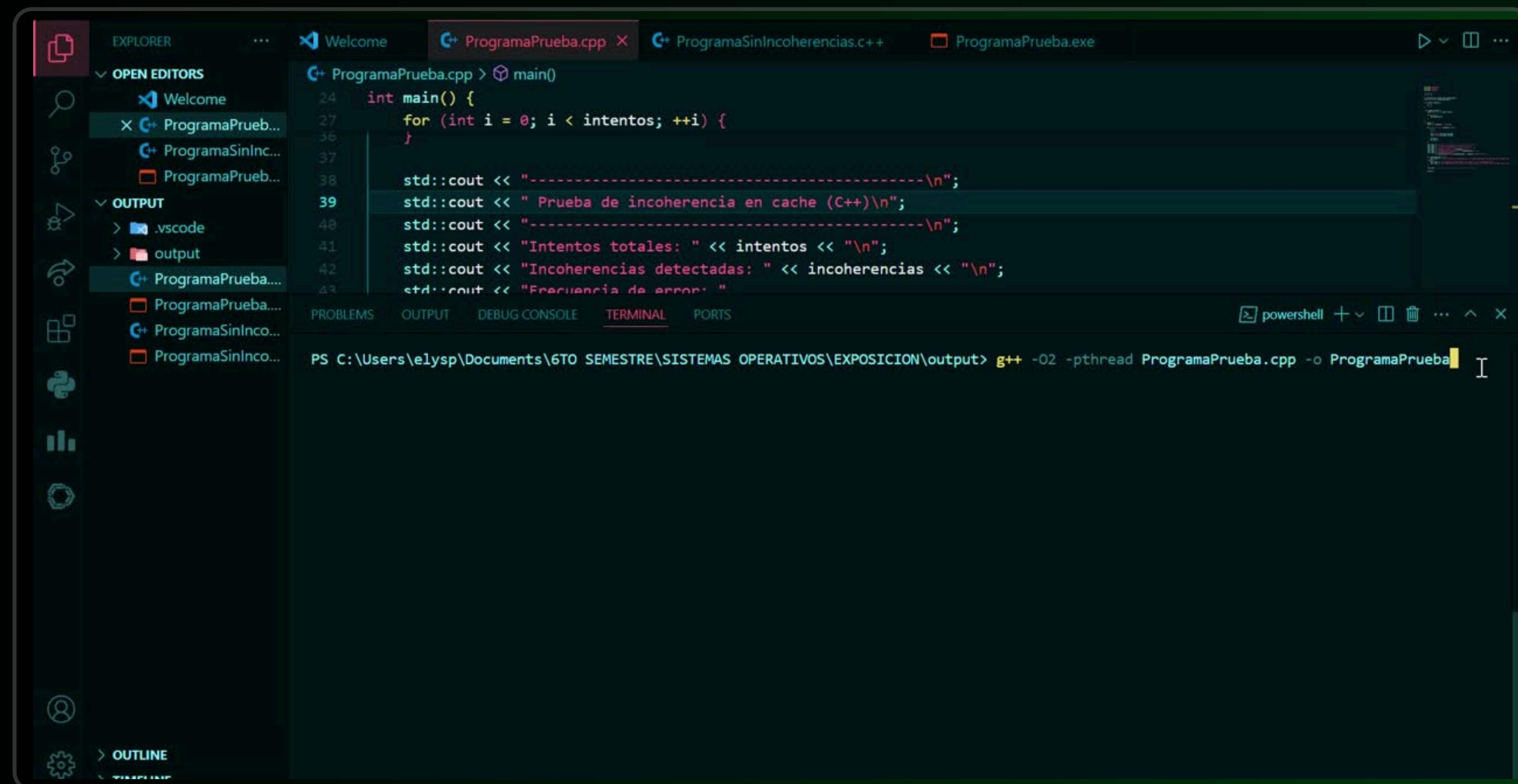
The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The left sidebar contains icons for Explorer, Search, Problems, and other extensions like Python and Timeline. The main area has tabs for 'Welcome', 'ProgramaPrueba.cpp', 'ProgramaSinIncoherencias.cpp', and 'ProgramaPrueba.exe'. The 'ProgramaPrueba.cpp' tab is active, displaying the following C++ code:

```
int main() {
    for (int i = 0; i < intentos; ++i) {

        std::cout << "-----\n";
        std::cout << " Prueba de incoherencia en cache (C++)\n";
        std::cout << "-----\n";
        std::cout << "Intentos totales: " << intentos << "\n";
        std::cout << "Incoherencias detectadas: " << incoherencias << "\n";
        std::cout << "Frecuencia de errores: " << errores << "\n";
    }
}
```

The terminal at the bottom shows the command: PS C:\Users\elysp\Documents\6TO SEMESTRE\SISTEMAS OPERATIVOS\EXPOSICION\output> g++ -O2 -pthread ProgramaPrueba.cpp -o ProgramaPrueba





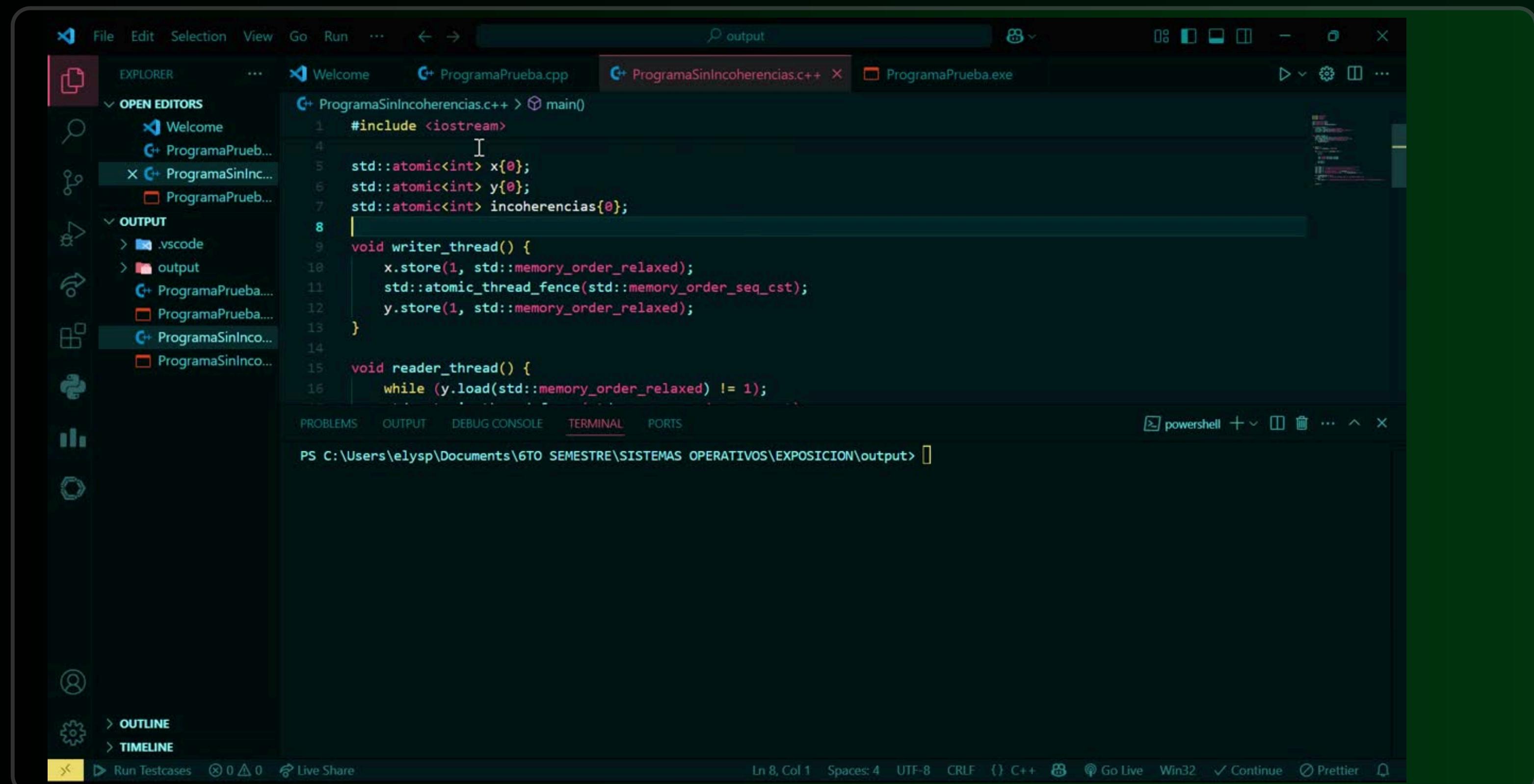
The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running on a Windows operating system. The main window displays a C++ file named `ProgramaPrueba.cpp`. The code in the editor is as follows:

```
int main() {
    for (int i = 0; i < intentos; ++i) {

    std::cout << "-----\n";
    std::cout << " Prueba de incoherencia en cache (C++)\n";
    std::cout << "-----\n";
    std::cout << "Intentos totales: " << intentos << "\n";
    std::cout << "Incoherencias detectadas: " << incoherencias << "\n";
    std::cout << "Frecuencia de errores: " <<
```

The terminal at the bottom shows the command used to compile the program:

```
PS C:\Users\elysp\Documents\6TO SEMESTRE\SISTEMAS OPERATIVOS\EXPOSICION\output> g++ -O2 -pthread ProgramaPrueba.cpp -o ProgramaPrueba
```



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The central editor window displays a C++ program named `ProgramaSinIncoherencias.c++`. The code implements a producer-consumer pattern with two threads: `writer_thread()` and `reader_thread()`. The `writer_thread()` function writes the value 1 to variables `x` and `y` using `std::memory_order_relaxed`. The `reader_thread()` function reads the value from `y` until it is no longer `1`. The code uses `std::atomic` and `std::atomic_thread_fence` to ensure thread safety.

```
#include <iostream>
#include <atomic>

std::atomic<int> x{0};
std::atomic<int> y{0};
std::atomic<int> incoherencias{0};

void writer_thread() {
    x.store(1, std::memory_order_relaxed);
    std::atomic_thread_fence(std::memory_order_seq_cst);
    y.store(1, std::memory_order_relaxed);
}

void reader_thread() {
    while (y.load(std::memory_order_relaxed) != 1);
}
```





The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The left sidebar contains icons for Explorer, Search, Problems, Outline, and Timeline. The Explorer panel shows open files: 'Welcome' (closed), 'ProgramaPrueba.cpp' (closed), 'ProgramaSinIncoherencias.c++' (active), and 'ProgramaPrueba.exe'. The OUTPUT panel shows '.vscode', 'output', and build logs for 'ProgramaPrueba....' and 'ProgramaSinInco...'.

The main editor area displays the following C++ code:

```
#include <iostream>
#include <atomic>

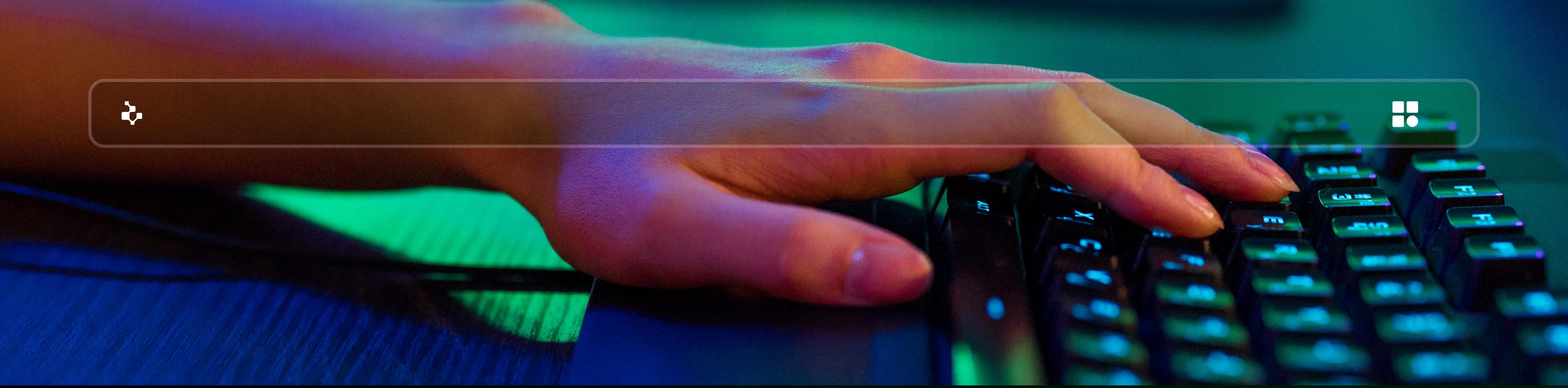
std::atomic<int> x{0};
std::atomic<int> y{0};
std::atomic<int> incoherencias{0};

void writer_thread() {
    x.store(1, std::memory_order_relaxed);
    std::atomic_thread_fence(std::memory_order_seq_cst);
    y.store(1, std::memory_order_relaxed);
}

void reader_thread() {
    while (y.load(std::memory_order_relaxed) != 1);
```

The terminal at the bottom shows the command prompt: PS C:\Users\elysp\Documents\6TO SEMESTRE\SISTEMAS OPERATIVOS\EXPOSICION\output>.





THANK YOU

