import java.util.concurrent.locks.ReentrantLock;

public class DeadlockExample {

private final ReentrantLock lock = new ReentrantLock();

public void process() {

if (lock.isHeldByCurrentThread()) {

// Nếu cùng một luồng gọi lại, tránh deadlock bằng cách không khóa nữa

System.out.println("Hàm đã được gọi trong luồng này, tiếp tục...");

return;

}

lock.lock();

try {

System.out.println("Đang xử lý...");

// Gọi lại chính nó để kiểm tra deadlock

process();

} finally {

lock.unlock();

}

}

public static void main(String[] args) {

DeadlockExample example = new DeadlockExample();

example.process();

}

}

**2. Dùng tryLock() để tránh chờ vô hạn**

Nếu một luồng không thể lấy được khóa, nó có thể bỏ qua hoặc đợi trong một khoảng thời gian thay vì bị kẹt mãi.

if (lock.tryLock(1, TimeUnit.SECONDS)) {

try {

process();

} finally {

lock.unlock();

}

} else {

System.out.println("Không thể lấy khóa, bỏ qua...");

}

**3. Dùng synchronized với kiểm tra recursion**

Nếu bạn dùng synchronized, có thể kiểm tra xem hàm đã được gọi trong cùng một luồng chưa.

Ví dụ:

java

CopyEdit

public class DeadlockExample {

private Thread owner = null;

public synchronized void process() {

if (Thread.currentThread() == owner) {

System.out.println("Hàm đang được xử lý bởi luồng này, tiếp tục...");

return;

}

owner = Thread.currentThread();

try {

System.out.println("Đang xử lý...");

process(); // Gọi đệ quy

} finally {

owner = null;

}

}

}

🔹 **Giải pháp:** Dùng biến owner để xác định luồng nào đang giữ quyền truy cập.

**4. Dùng Queue để xử lý yêu cầu thay vì gọi đệ quy**

Thay vì gọi lại hàm trong cùng một luồng, có thể đưa yêu cầu vào hàng đợi và xử lý lần lượt.

Ví dụ dùng BlockingQueue:

java

CopyEdit

import java.util.concurrent.\*;

public class TaskProcessor {

private final BlockingQueue<Runnable> queue = new LinkedBlockingQueue<>();

private final Thread worker;

public TaskProcessor() {

worker = new Thread(() -> {

while (true) {

try {

Runnable task = queue.take();

task.run();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

break;

}

}

});

worker.start();

}

public void process() {

queue.offer(() -> {

System.out.println("Đang xử lý tác vụ...");

});

}

}

🔹 **Giải pháp:** Các yêu cầu được xử lý tuần tự, tránh deadlock.

**5. Dùng Async để tránh gọi lặp lại trong cùng một luồng**

Nếu có thể, hãy xử lý yêu cầu không đồng bộ thay vì gọi lại trực tiếp.

Ví dụ dùng ExecutorService:

java

CopyEdit

import java.util.concurrent.\*;

public class AsyncProcessor {

private final ExecutorService executor = Executors.newSingleThreadExecutor();

public void process() {

executor.submit(() -> {

System.out.println("Đang xử lý tác vụ...");

});

}

}

🔹 **Giải pháp:** Đảm bảo các yêu cầu không bị gọi lại trong cùng một luồng, tránh deadlock.

**🎯 Tổng kết**

| **Cách** | **Khi nào dùng?** | **Lợi ích** |
| --- | --- | --- |
| ReentrantLock | Khi cần tái nhập khóa | Tránh deadlock nếu gọi lại trong cùng luồng |
| tryLock() | Khi không muốn chờ vô hạn | Tránh chặn luồng |
| synchronized + owner | Khi chỉ có một luồng chính gọi lại | Tránh đệ quy vô hạn |
| Queue (BlockingQueue) | Khi có nhiều yêu cầu đến cùng lúc | Xử lý tuần tự, tránh deadlock |
| Async (ExecutorService) | Khi có thể xử lý bất đồng bộ | Tách biệt luồng, tránh xung đột |

Bạn đang gặp vấn đề cụ thể với deadlock trong dự án nào không? Mình có thể giúp debug code cho bạn! 🚀

**1. Kiểm tra queueInputBuffer() có hoạt động đúng không**

Thêm log khi queueInputBuffer() được gọi:

java

CopyEdit

Log.d("EncoderFilter", "queueInputBuffer() frame: " + frameIndex);

encoder.queueInputBuffer(inputBufferIndex, 0, bufferSize, presentationTimeUs, flags);

Nếu queueInputBuffer() không chạy sau frame 60, có thể dữ liệu từ Startrail bị gián đoạn.

**✅ 2. Giải phóng outputBuffer đúng cách**

Sau khi dequeueOutputBuffer() trả về một index hợp lệ, phải gọi releaseOutputBuffer().  
Nếu thiếu bước này, encoder sẽ **đầy buffer** và -1 xuất hiện liên tục.

java

CopyEdit

if (outputBufferIndex >= 0) {

encoder.releaseOutputBuffer(outputBufferIndex, false);

Log.d("EncoderFilter", "Released outputBuffer: " + outputBufferIndex);

}

Nếu bạn không thấy log này sau frame 60, có thể Encoder không giải phóng output buffer.

**✅ 3. Kiểm tra Muxer có đang nhận dữ liệu từ Encoder không**

Nếu muxer.writeSampleData() không được gọi, dữ liệu không được lấy ra khỏi encoder, gây deadlock.

java

CopyEdit

if (muxerStarted) {

muxer.writeSampleData(trackIndex, encodedData, bufferInfo);

Log.d("Muxer", "Wrote sample data, size: " + bufferInfo.size);

}

Nếu log này dừng ở frame 60, nghĩa là muxer không tiêu thụ dữ liệu.

**✅ 4. Tăng kích thước buffer giữa Startrail và Encoder**

Nếu **Startrail bị chậm**, Encoder có thể không nhận đủ dữ liệu.  
Bạn có thể thử tăng BufferChannel capacity lên **2 hoặc 3** để tránh tắc nghẽn.

java

CopyEdit

BufferChannel encoderBuffer = new BufferChannel(3); // Tăng buffer lên 3 giúp giảm nghẽn

Điều này giúp Startrail có chỗ đẩy frame mà không bị chặn.

**✅ 5. Xử lý EOS đúng cách để tránh Encoder chờ vô tận**

Nếu EOS không được đánh dấu đúng cách, Encoder có thể **chờ mãi mãi** mà không kết thúc.  
Hãy kiểm tra xem có đánh dấu **End Of Stream (EOS)** không:

java

CopyEdit

if (isEndOfStream) {

encoder.signalEndOfInputStream();

Log.d("EncoderFilter", "End Of Stream signaled");

}

Nếu dequeueOutputBuffer() bị -1 liên tục **vì EOS**, hãy thử gửi một frame trống cuối cùng.

**🚀 Tóm tắt cách khắc phục**

| **Vấn đề** | **Giải pháp** |
| --- | --- |
| queueInputBuffer() không chạy sau frame 60 | Kiểm tra dữ liệu từ Startrail có đến Encoder không |
| releaseOutputBuffer() không được gọi | Thêm log khi giải phóng buffer |
| muxer.writeSampleData() không chạy | Kiểm tra Muxer có bị kẹt không |
| Pipeline bị tắc giữa Startrail & Encoder | Tăng BufferChannel capacity lên 2-3 |
| Encoder không kết thúc do EOS | Gửi signalEndOfInputStream() đúng cách |

Bạn thử debug và gửi thêm log nếu vẫn bị lỗi nhé! 🚀

Nếu ReadBufferChannel.send() được gọi **hai lần trước khi lần đầu tiên xử lý xong**, điều này có thể gây **ghi đè dữ liệu hoặc deadlock**, đặc biệt nếu BufferChannel có **capacity = 1**.

**🔥 Vấn đề có thể gặp:**

1. **Dữ liệu mới ghi đè lên dữ liệu cũ**
   * Nếu send() được gọi hai lần nhưng buffer chỉ có **capacity = 1**, dữ liệu đầu tiên có thể **bị ghi đè** trước khi được xử lý.
2. **Deadlock nếu Startrail chưa đọc xong**
   * Nếu Decoder tiếp tục gọi send(), nhưng Startrail chưa đọc xong frame trước đó, pipeline có thể bị **tắc nghẽn**.
3. **Race condition nếu send() chạy trên nhiều thread**
   * Nếu send() không được đồng bộ hóa đúng cách, hai luồng có thể ghi dữ liệu cùng lúc, gây lỗi.

**🛠️ Cách khắc phục**

**✅ 1. Chặn send() nếu buffer chưa xử lý xong**

Thêm kiểm tra **trước khi gửi dữ liệu mới**:

java

CopyEdit

synchronized (this) {

while (!buffer.isEmpty()) {

Log.d("ReadBufferChannel", "Buffer chưa xử lý xong, chờ...");

wait(); // Chờ đến khi buffer trống

}

buffer.write(newData);

notifyAll(); // Báo cho luồng đang chờ đọc dữ liệu

}

**Lợi ích:**

* Ngăn send() ghi đè dữ liệu khi buffer chưa được xử lý xong.
* Tránh race condition nếu nhiều luồng gọi send() cùng lúc.

**✅ 2. Kiểm tra trySend() để tránh block vô hạn**

Thay vì chờ, bạn có thể **bỏ qua frame nếu buffer đầy** bằng cách dùng trySend():

java

CopyEdit

if (!buffer.tryWrite(newData)) {

Log.w("ReadBufferChannel", "Bỏ qua frame do buffer đầy");

}

**Lợi ích:**

* Tránh deadlock nếu Startrail bị chậm.
* Giúp pipeline tiếp tục chạy mà không bị tắc nghẽn.

**✅ 3. Tăng buffer size để giảm tắc nghẽn**

Nếu pipeline bị tắc khi **Startrail xử lý chậm**, thử tăng BufferChannel capacity lên **2 hoặc 3**:

java

CopyEdit

BufferChannel readBuffer = new BufferChannel(3); // Tăng buffer size để tránh chặn

**Lợi ích:**

* Cho phép pipeline tiếp tục chạy mà không bị block ngay lập tức.

**✅ 4. Đảm bảo notify() sau khi đọc xong**

Trong Startrail, khi frame được xử lý xong, gọi notify() để **báo cho send() rằng buffer đã trống**:

java

CopyEdit

synchronized (this) {

buffer.read();

notifyAll(); // Báo cho send() tiếp tục

}

**Lợi ích:**

* Giúp send() không bị block quá lâu.
* Đảm bảo dữ liệu không bị ghi đè trước khi đọc xong.

**🚀 Tóm tắt cách khắc phục**

| **Vấn đề** | **Giải pháp** |
| --- | --- |
| send() gọi hai lần trước khi xử lý xong | Dùng synchronized + wait/notify để tránh ghi đè |
| Buffer bị đầy, gây tắc nghẽn | Dùng trySend() để bỏ qua frame nếu cần |
| Pipeline bị chậm do buffer nhỏ | Tăng buffer size lên 2-3 |
| Startrail không báo send() tiếp tục | Gọi notifyAll() sau khi đọc xong |

Bạn kiểm tra thử log và áp dụng cách trên nhé! 🚀

4o

**You said:**

decoderFilter chạy nhanh hơn startrail

**ChatGPT said:**

Nếu **decoderFilter chạy nhanh hơn startrail**, pipeline có thể bị **tắc nghẽn** vì:

* **decoderFilter liên tục đẩy dữ liệu**, nhưng startrail chưa xử lý xong.
* **BufferChannel đầy**, khiến decoderFilter phải chờ hoặc ghi đè dữ liệu.
* **Deadlock xảy ra**, nếu decoderFilter chờ BufferChannel trống nhưng startrail chưa xong.

**🛠️ Cách xử lý khi decoderFilter chạy nhanh hơn startrail**

**✅ 1. Chặn decoderFilter nếu startrail chưa xử lý xong**

Hãy **đồng bộ giữa decoderFilter và startrail** bằng Semaphore hoặc BlockingQueue.

**🔹 Cách 1: Dùng Semaphore để giới hạn tốc độ**

Cho phép decoder chỉ đẩy dữ liệu khi **Startrail đã xử lý xong**.

java

CopyEdit

Semaphore semaphore = new Semaphore(1); // Chỉ cho phép 1 buffer tồn tại

// Trong Decoder

semaphore.acquire(); // Chờ Startrail xử lý xong

bufferChannel.send(decodedFrame);

semaphore.release(); // Giải phóng khi Startrail xong

// Trong Startrail

frame = bufferChannel.receive();

process(frame);

semaphore.release(); // Cho phép Decoder tiếp tục

**🔹 Lợi ích:**

* Giúp Decoder **chỉ đẩy dữ liệu khi Startrail đã xong**.
* Tránh buffer bị **đầy** hoặc **ghi đè dữ liệu**.

**✅ 2. Tăng buffer giữa Decoder và Startrail**

Nếu **Startrail chậm nhưng vẫn hoạt động bình thường**, hãy thử **tăng kích thước buffer**:

java

CopyEdit

BufferChannel bufferChannel = new BufferChannel(3); // Cho phép 3 frame chờ xử lý

**🔹 Lợi ích:**

* Giúp pipeline không bị tắc ngay lập tức khi Startrail chậm.
* Giảm áp lực lên Decoder.

**✅ 3. Giới hạn FPS của Decoder**

Nếu decoderFilter quá nhanh, bạn có thể **thêm delay** để chờ startrail theo kịp:

java

CopyEdit

Thread.sleep(10); // Chờ 10ms trước khi decode frame tiếp theo

**🔹 Lợi ích:**

* Giảm tốc độ Decoder mà không ảnh hưởng pipeline.
* Giúp Startrail có đủ thời gian xử lý.

**✅ 4. Xử lý ưu tiên trong Thread để tránh deadlock**

Nếu decoderFilter và startrail chạy trên **các thread khác nhau**, hãy dùng **ExecutorService** để điều phối:

java

CopyEdit

ExecutorService executor = Executors.newFixedThreadPool(2);

executor.submit(() -> decoder.decode());

executor.submit(() -> startrail.process());

**🔹 Lợi ích:**

* Tránh việc **Decoder chiếm quá nhiều CPU**, giúp Startrail có tài nguyên để xử lý.

**✅ 5. Kiểm tra dequeueOutputBuffer() để tránh buffer bị kẹt**

Nếu Encoder bị nghẽn, hãy đảm bảo nó đang **đọc dữ liệu từ Startrail đúng cách**:

java

CopyEdit

if (encoder.dequeueOutputBuffer(bufferInfo, TIMEOUT\_US) >= 0) {

encoder.releaseOutputBuffer(outputBufferIndex, false);

}

Nếu dequeueOutputBuffer() bị -1 liên tục, **Encoder có thể không nhận dữ liệu từ Startrail**, gây tắc nghẽn.

**🔥 Tóm tắt cách khắc phục**

| **Vấn đề** | **Giải pháp** |
| --- | --- |
| **Decoder chạy nhanh hơn Startrail** | Dùng Semaphore để chặn Decoder |
| **Buffer đầy, gây nghẽn** | Tăng buffer lên 2-3 |
| **Decoder quá nhanh** | Thêm Thread.sleep(10) |
| **Thread không đồng bộ** | Dùng ExecutorService để điều phối |
| **Encoder không đọc dữ liệu** | Đảm bảo dequeueOutputBuffer() hoạt động |

Bạn thử áp dụng các cách trên và gửi thêm log nếu vẫn gặp lỗi nhé! 🚀