Sincronización con MappedByteBuffer

1. ¿Qué es MappedByteBuffer?

MappedByteBuffer es una clase en Java que permite mapear un archivo en memoria para que se pueda acceder a su contenido como si fuera un array de bytes. Es útil para compartir datos entre procesos porque cualquier cambio que se realice en el buffer se refleja en el archivo subyacente, y viceversa. De esta forma, múltiples procesos pueden acceder a la misma región de memoria compartida a través del mismo archivo.

2. ¿Cómo funciona la sincronización?

En este ejemplo, se usan dos procesos diferentes:

- **Proceso 1 (Escritor)**: Escribe datos en la memoria compartida.
- **Proceso 2 (Lector)**: Lee los datos desde la memoria compartida cuando detecta que el Proceso 1 ha terminado de escribir.

Ambos procesos acceden a la misma región de memoria compartida mapeada a un archivo llamado shared_memory.dat. El Proceso 1 escribe un mensaje en el archivo y el Proceso 2 lee ese mensaje una vez que detecta que hay contenido disponible.

3. Código Explicado

Proceso 1: Escritor (SharedMemoryWriter)

Este proceso escribe un mensaje en la memoria compartida y establece un indicador (flag) para que el lector sepa que hay un mensaje listo para ser leído.

```
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class SharedMemoryWriter {

   private static final String FILE_PATH = "shared_memory.dat";

   public static void main(String[] args) {
      try {
         // Abrir o crear el archivo de memoria compartida con permisos de
lectura/escritura
         RandomAccessFile file = new RandomAccessFile(FILE_PATH, "rw");
         FileChannel fileChannel = file.getChannel();

         // Mapear 1024 bytes del archivo a memoria compartida
         MappedByteBuffer buffer =
fileChannel.map(FileChannel.MapMode.READ_WRITE, 0, 1024);

         // Escribir un mensaje en el buffer compartido (carácter por carácter)
         String message = "Mensaje del proceso escritor";
         for (int i = 0; i < message.length(); i++) {
            buffer.put(i, (byte) message.charAt(i));
         }
```

```
        // Añadir un byte de control al final del mensaje indicando que el mensaje está
listo
        buffer.put(message.length(), (byte) 1); // Este byte actúa como un flag o marcador

        System.out.println("Mensaje escrito en memoria compartida.");

        // Cerrar el archivo y el canal
        fileChannel.close();
        file.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

Explicación del Proceso 1 (Escritor):

- Abre el archivo shared_memory.dat en modo lectura/escritura. Si el archivo no existe, se crea.
- Mapea 1024 bytes del archivo en un MappedByteBuffer, creando un área de memoria compartida.
- Escribe un mensaje carácter por carácter en el buffer.
- Añade un byte adicional al final del mensaje (buffer.put(message.length(), (byte) 1);) para indicar que el mensaje ha sido escrito correctamente. Este byte funciona como un "flag" (bandera) para que el proceso de lectura sepa que el mensaje está listo.
- Cierra el canal y el archivo.

Proceso 2: Lector (SharedMemoryReader)

Este proceso lee los datos de la memoria compartida y espera hasta que detecta el byte de control que indica que el mensaje está listo.

```
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class SharedMemoryReader {

  private static final String FILE_PATH = "shared_memory.dat";

  public static void main(String[] args) {
    try {
      // Abrir el archivo de memoria compartida en modo lectura/escritura
      RandomAccessFile file = new RandomAccessFile(FILE_PATH, "rw");
      FileChannel fileChannel = file.getChannel();

      // Mapear 1024 bytes del archivo a memoria compartida
```

```
        MappedByteBuffer buffer =
fileChannel.map(FileChannel.MapMode.READ_WRITE, 0, 1024);

        // Esperar hasta que el byte de control indique que el mensaje está listo
        System.out.println("Esperando a que el mensaje esté disponible...");
        while (buffer.get(100) == 0) {  // Comprobamos si el byte de control tiene el valor 1
          Thread.sleep(500); // Espera medio segundo antes de verificar de nuevo
        }

        // Leer el mensaje desde la memoria compartida
        StringBuilder message = new StringBuilder();
        for (int i = 0; i < 1024 && buffer.get(i) != 0; i++) {
          message.append((char) buffer.get(i));
        }

        System.out.println("Mensaje leído: " + message.toString());

        // Cerrar el archivo y el canal
        fileChannel.close();
        file.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

Explicación del Proceso 2 (Lector):

- Abre el mismo archivo shared_memory.dat en modo lectura/escritura.
- Mapea 1024 bytes del archivo en un MappedByteBuffer.
- Espera en un bucle hasta que el byte de control (ubicado en buffer.get(100)) se establezca en 1, lo que indica que el mensaje ha sido escrito por el proceso 1.
- Una vez detectado el byte de control, lee el mensaje desde el buffer compartido.
- Cierra el canal y el archivo.

4. ¿Cómo se comunican los procesos?

- Ambos procesos acceden al mismo archivo shared_memory.dat.
- El proceso de escritura mapea este archivo en memoria y escribe un mensaje.
- El proceso de lectura mapea el mismo archivo en memoria y monitorea un byte de control para saber cuándo leer el mensaje.
- De esta manera, ambos procesos se comunican sin usar hilos ni sockets, sino usando un archivo mapeado en memoria que funciona como un área de memoria compartida.

5. Demostración del uso

Sincronización con MappedByteBuffer

Para ver este ejemplo en acción:

1. **Compila ambos archivos Java**:

```
javac SharedMemoryWriter.java
javac SharedMemoryReader.java
```

2. **Ejecútalos en terminales diferentes**:

- En la primera terminal, ejecuta el proceso de escritura:

```
java SharedMemoryWriter
```

- En la segunda terminal, ejecuta el proceso de lectura:

```
java SharedMemoryReader
```

El proceso de lectura esperará hasta que el proceso de escritura haya escrito el mensaje en la memoria compartida y lo haya marcado como listo (estableciendo el byte de control a 1. Una vez que detecte el mensaje, lo leerá y lo mostrará en la consola.