Progrmación Concurrente, 23 -I, CG01, June 8, 2023

Java Communicating Sequential Processes (JCSP)

Martínez Buenrostro Jorge Rafael.

Unidad Iztapalapa
Universidad Autónoma Metropolitana
*molap*96@*gmail.com*

Resumen: En este reporte se describirá y mostrará el uso de concurrencia de JCSP.
Palabras Clave: Concurrencia, Sincronización, Entidad Activa, Comunicación.

Introducción

La programación concurrente es una técnica que permite expresar y ejecutar tareas que pueden ocurrir simultáneamente, aprovechando los recursos de los sistemas multiprocesador o distribuidos. Sin embargo, la programación concurrente también implica desafíos como la sincronización, la comunicación, la exclusión mutua y la detección de bloqueos entre los procesos concurrentes. Para abordar estos desafíos, se han propuesto diversos modelos y lenguajes de programación concurrente, basados en diferentes paradigmas y abstracciones.
Uno de estos modelos es el de Communicating Sequential Processes (CSP), propuesto por Tony Hoare en 1978, que se basa en el paso de mensajes sincrónicos a través de canales entre procesos secuenciales que pueden combinarse mediante operadores algebraicos. El modelo CSP tiene una semántica matemática definida y permite verificar propiedades de concurrencia de los sistemas, como la confidencialidad, la autenticación y la ausencia de bloqueos.
JCSP (Communicating Sequential Processes for Java) es una librería de concurrencia desarrollada específicamente para el lenguaje de programación Java. Se basa en el modelo CSP (Communicating Sequential Processes) y proporciona una implementación de dicho modelo.

Descripción general del lenguaje

Java es un lenguaje de programación orientado a objetos, compilado e interpretado, que ofrece soporte nativo para la programación concurrente mediante el uso de hilos (threads). Un hilo es una unidad básica de ejecución que puede compartir memoria y recursos con otros hilos dentro del mismo proceso. Java provee una clase Thread que encapsula el comportamiento y el estado de un hilo, así como una interfaz Runnable que define el método run que debe ejecutar el hilo. Además, Java ofrece mecanismos

de sincronización basados en monitores, como el uso de la palabra clave synchronized para crear bloques o métodos críticos, o el uso de los métodos wait, notify y notifyAll para coordinar la espera y la notificación entre hilos.
Sin embargo, el uso directo de hilos y monitores en Java puede resultar complejo, propenso a errores y difícil de razonar. Por ello, se han desarrollado diversas bibliotecas y extensiones que ofrecen abstracciones de más alto nivel para facilitar la programación concurrente en Java. Algunas de estas bibliotecas son java.util.concurrent, que provee utilidades como ejecutores, colas bloqueantes, semáforos, cerrojos y barreras; java.util.stream, que permite crear flujos paralelos de datos; o JCSP, que implementa el modelo CSP para Java.

Descripción general del modelo de concurrencia

JCSP es una implementación del modelo CSP para el lenguaje Java, desarrollada por Peter Welch y otros colaboradores en la Universidad de Kent. JCSP es una biblioteca que ofrece las primitivas y las herramientas necesarias para construir sistemas concurrentes en Java siguiendo el modelo CSP. JCSP no requiere un conocimiento profundo del sistema matemático subyacente a CSP, sino que permite que los programadores puedan lograr software bien comportado siguiendo reglas simples. Este modelos se basa en el paso de mensajes a través de canales entre procesos secuenciales que pueden combinarse mediante operadores de elección, secuencia y paralelismo.
Un canal es un medio de comunicación entre procesos secuenciales que permite la transmisión de mensajes de forma sincrónica. Los canales pueden ser nombrados o anónimos, y admiten distintos tipos de datos. Su propósito principal es la sincronización y coordinación de acciones entre procesos que comparten eventos.

JCSP se basa en la noción de procesos como componentes comunicantes que interactúan exclusivamente mediante primitivas de sincronización basadas en CSP, tales como canales (One2OneChannel, Any2OneChannel, One2AnyChannel, Any2AnyChannel), canales con llamada (One2OneCallChannel), temporizadores (CSTimer), equipos (Crew), barreras (Barrier), barreras alternativas (AltingBarrier), cubetas (Bucket), u otros modos bien definidos de acceso a objetos pasivos compartidos. Los procesos no invocan los métodos unos de otros.
Los procesos en JCSP se implementan mediante la interfaz CSProcess, que define el método run que debe ejecutar el proceso. Los procesos pueden ser creados y ejecutados mediante la clase Parallel, que recibe un arreglo de procesos y los ejecuta concurrentemente. Los procesos pueden comunicarse mediante canales, que son objetos que implementan las interfaces ChannelInput y ChannelOutput, que definen los métodos read y write, respectivamente. Los canales pueden ser sincrónicos o con buffer, y pueden tener diferentes tipos de datos. Los canales se usan para sincronizar y coordinar las acciones de los procesos que comparten eventos.
Los procesos en JCSP pueden elegir entre diferentes eventos de entrada usando la clase Alternative, que recibe un arreglo de objetos Guard, que representan condiciones de habilitación para los eventos. Los objetos Guard pueden ser canales, temporizadores, barreras o condiciones personalizadas. La clase Alternative ofrece

el método select, que devuelve el índice del evento que se ha producido, o el método priSelect, que devuelve el índice del evento con mayor prioridad que se ha producido.

JCSP ofrece también otras herramientas para facilitar la construcción de sistemas concurrentes, como redes de procesos, procesos prefabricados, procesos móviles, extensiones gráficas y de red, y soporte para la verificación formal de propiedades de concurrencia.
JCSP proporciona abstracciones para crear procesos concurrentes y canales de comunicación. Los procesos se implementan como clases Java que extienden la interfaz CSProcess. Los canales se definen mediante interfaces específicas, como ChannelInput y ChannelOutput, que permiten la comunicación entre los procesos.

Soporte para la creación de EA

JCSP ofrece soporte para la creación de entidades activas (EA), que son componentes concurrentes que encapsulan su estado y su comportamiento, y que se comunican mediante paso de mensajes. Las EA en JCSP se implementan mediante procesos que usan canales para interactuar con otros procesos o con el entorno externo.
Un ejemplo de EA en JCSP es el proceso Bucket, que representa una cubeta con una capacidad limitada de elementos. El proceso Bucket tiene dos canales de entrada: uno para recibir elementos que se quieren depositar en la cubeta, y otro para recibir peticiones de elementos que se quieren extraer de la cubeta. El proceso Bucket tiene también un canal de salida: uno para enviar elementos que se han extraído de la cubeta. El proceso Bucket mantiene un estado interno que consiste en una cola con los elementos almacenados en la cubeta y un contador con el número de elementos disponibles. El proceso Bucket ejecuta un ciclo infinito en el que espera por eventos de entrada o salida, y actualiza su estado según corresponda.

Otro ejemplo de EA en JCSP es el proceso AltingBarrier, que representa una barrera alternativa que puede sincronizar a un conjunto variable de procesos. El proceso AltingBarrier tiene dos canales de entrada: uno para recibir peticiones de inscripción o cancelación de procesos en la barrera, y otro para recibir peticiones de activación o desactivación de la barrera. El proceso AltingBarrier tiene también un canal de salida: uno para enviar señales a los procesos inscritos cuando la barrera se activa. El proceso AltingBarrier mantiene un estado interno que consiste en un conjunto con los procesos inscritos en la barrera y un booleano con el estado de activación de la barrera. El proceso AltingBarrier ejecuta un ciclo infinito en el que espera por eventos de entrada o salida, y actualiza su estado según corresponda.

Para crear un Elemento Activo en JCSP, se define una clase que implementa la interfaz Active. Esta interfaz proporciona métodos para la inicialización, ejecución y finalización del Elemento Activo. El proceso de ejecución se realiza en paralelo con otros procesos y se sincroniza mediante canales para lograr una comunicación segura.

Modelos de comunicación que soporta

JCSP soporta principalmente el modelo de comunicación por paso de mensajes sincrónicos entre procesos secuenciales, siguiendo el modelo CSP. Este modelo implica que los procesos solo pueden comunicarse mediante el envío y la recepción de mensajes a través de canales, y que cada mensaje requiere la participación simultánea del emisor y del receptor. Este modelo garantiza la exclusión mutua entre los procesos comunicantes y evita las condiciones de carrera sobre los datos compartidos. Dependiendo del tipo y las propiedades de los canales utilizados. Algunos ejemplos son:

- **Comunicación síncrona:** Los procesos se sincronizan en los canales, de modo que el emisor se bloquea hasta que el receptor esté listo para recibir el mensaje.
- **Comunicación por eventos:** JCSP también admite la comunicación basada en eventos, donde los procesos pueden enviar y recibir eventos para coordinar su ejecución.
- **Comunicación punto a punto:** Se utiliza un canal unidireccional entre dos procesos, donde uno envía y otro recibe mensajes. El envío y la recepción son operaciones bloqueantes, es decir, el emisor espera hasta que el receptor esté listo para recibir el mensaje, y viceversa.
- **Comunicación broadcast:** Se utiliza un canal con múltiples lectores y un solo escritor, donde el escritor envía un mensaje a todos los lectores simultáneamente. El envío es una operación no bloqueante, es decir, el escritor no espera a que los lectores reciban el mensaje.
- **Comunicación any-to-any:** Se utiliza un canal con múltiples escritores y múltiples lectores, donde cualquier proceso puede enviar o recibir mensajes a cualquier otro proceso conectado al canal. El envío y la recepción son operaciones bloqueantes con selección aleatoria, es decir, el emisor espera hasta que haya al menos un receptor disponible, y viceversa, y se selecciona uno al azar para realizar el intercambio.
- **Comunicación con capacidad:** Se utiliza un canal con capacidad limitada o infinita, donde los mensajes se almacenan en una cola interna hasta que son consumidos por los receptores. El envío y la recepción son operaciones bloqueantes o no bloqueantes, dependiendo de si la cola está llena o vacía, respectivamente.
- **Comunicación con objetos o primitivas:** Se utiliza un canal que permite el envío de objetos o tipos primitivos de Java, como enteros, booleanos o caracteres. Los objetos se transmiten por referencia o por valor, dependiendo de si el canal es seguro o inseguro para la transmisión de objetos.

Herramientas de sincronización

JCSP proporciona diversas herramientas de sincronización para facilitar la coordinación y la comunicación entre los procesos concurrentes. Algunas de estas herramientas son:

- **Semáforos:** Son objetos que permiten a un proceso adquirir o liberar un recurso compartido, es decir, esperar hasta que haya un recurso disponible para continuar su ejecución, o notificar que ha terminado de usar el recurso. Los semáforos pueden ser binarios o contadores, dependiendo de si el recurso es único o múltiple, respectivamente.
- **Barreras:** Son objetos que permiten a un conjunto de procesos sincronizarse en un punto común, es decir, esperar hasta que todos los procesos lleguen a la barrera para continuar su ejecución. Las barreras pueden ser locales o distribuidas, dependiendo de si los procesos que participan están en la misma máquina o en distintas máquinas, respectivamente.
- **Alternativas:** Son objetos que permiten a un proceso esperar por uno o más eventos de comunicación, como el envío o la recepción de un mensaje, y seleccionar uno de ellos para continuar su ejecución. Las alternativas pueden ser deterministas o no deterministas, dependiendo de si el criterio de selección es fijo o aleatorio, respectivamente.
- **AltingBarrier:** Esta herramienta combina una barrera con una espera alternativa (alt), lo que permite a los procesos esperar en la barrera o realizar otras acciones según las condiciones de comunicación.

Ejemplo mínimo

A continuación se muestra un ejemplo mínimo de cómo utilizar JCSP para crear dos procesos que se comunican entre sí:

Listing 1: Ejemplo JCSP

```
public static void main(String[] args)
{
  //Crear los canales de comunicacion
  Any2OneChannel<String> canal1=Channel.any2one();
  Any2OneChannel<String> canal2=Channel.any2one();

  // Crear los procesos
  CSProcess proceso1=new Proceso1(canal1.out(),canal2.in());
  CSProcess proceso2=new Proceso2(canal2.out(),canal1.in());

  // Ejecutar los procesos en paralelo
```

```
  Parallel parallel=new Parallel();
  parallel.addProcess(proceso1);
  parallel.addProcess(proceso2);
  parallel.run();
}

// Proceso 1
static class Proceso1 implements CSProcess
{
  private ChannelOutput<String> salida;
  private ChannelInput<String> entrada;

  public Proceso1(ChannelOutput<String> salida,
                                  ChannelInput<String> entrada)
  {
    this.salida=salida;
        this.entrada=entrada;
  }

  public void run()
  {
    // Enviar un mensaje al proceso 2
    salida.write("Hola desde Proceso 1");

    // Esperar la respuesta del proceso 2
    String respuesta = entrada.read();
    System.out.println("Proceso 1 recibi : " + respuesta);
  }
}

// Proceso 2
static class Proceso2 implements CSProcess
{
  private ChannelOutput<String> salida;
  private ChannelInput<String> entrada;

  public Proceso2(ChannelOutput<String> salida,
                                  ChannelInput<String> entrada)
  {
    this.salida=salida;
    this.entrada=entrada;
  }

  public void run()
  {
```

```
        // Esperar el mensaje del proceso 1
        String mensaje=entrada.read();

        System.out.println("Proceso 2 recibi : " + mensaje);

        // Responder al proceso 1
        salida.write("Hola desde Proceso 2");
    }
}
```

En este ejemplo, se crean dos canales de comunicación (canal1 y canal2) utilizando la clase Any2OneChannel de JCSP. Estos canales se utilizan para establecer la comunicación entre los dos procesos Proceso1 y Proceso2.

1. El Proceso1 envía el mensaje "Hola desde Proceso 1" al Proceso2 a través del canal1. Luego, espera a recibir la respuesta del Proceso2 a través del canal2.
2. El Proceso2 espera a recibir el mensaje del Proceso1 a través del canal2. Luego, muestra el mensaje recibido y responde al Proceso1 enviando "Hola desde Proceso 2" a través del canal1.

Al ejecutar el programa, verás la salida que muestra los mensajes intercambiados entre los dos procesos:

Listing 2: Salida en termianl

```
Proceso 2 recibi : Hola desde Proceso 1
Proceso 1 recibi : Hola desde Proceso 2
```

Conclusiones

JCSP ofrece una biblioteca sólida y eficiente para la programación concurrente en Java, basada en el modelo de concurrencia CSP. Permite crear procesos concurrentes y establecer una comunicación segura entre ellos mediante canales. Además, proporciona herramientas de sincronización y soporte para la creación de Elementos Activos. Utilizar JCSP puede mejorar la concurrencia y la eficiencia de las aplicaciones Java, evitando problemas comunes asociados con la concurrencia, como condiciones de carrera y bloqueos.
Por otro lado, es importante tener en cuenta que el modelo CSP es abstracto y matemático, por lo que no tiene en cuenta aspectos prácticos como el rendimiento, la escalabilidad, la tolerancia a fallos o la distribución de los procesos. Además, requiere que los procesos se comuniquen mediante canales sincrónicos, lo que puede generar bloqueos o ineficiencias si los procesos no están disponibles o tienen diferentes velocidades de ejecución. Es decir, CSP no proporciona soporte para la comunicación asíncrona, la movilidad de los procesos ni la dinámica de creación y terminación de los mismos. Esto puede resultar complejo o poco intuitivo para los programadores que están acostumbrados a otros paradigmas o lenguajes de programación concurrente.