Análisis, ventajas y desventajas de los modelos de concurrencia.

A lo largo de este curso se hemos trabajado con los cuatro modelos de concurrencia, realizando su aplicación dentro de nuestro proyecto de agregador de servicios. Cada uno de estos modelos ha presentado características y limitantes a la hora de implementarlos. En este reporte, se dará una explicación de cada modelo, así como un listado de ventajas y desventajas.

Antes de comenzar con cada modelo, es importante realizar la definición de un proceso, ya que en cierta manera hacen parte de las bases de la concurrencia. Un **proceso** es una instancia de un programa, el cual tiene una dirección virtual de memoria, librerías compartidas, datos de registro, entre otros.

Los sistemas operativos permiten a muchos procesos ejecutar concurrentemente, los procesos se intercambian rápidamente, alrededor de 20 ms toman estos cambios lo que genera en él usuario una impresión de paralelismo.

Treads and locks.

Los threads son secuencias de instrucciones manejadas por un scheduler, múltiples threads pueden existir dentro de un solo proceso. Los threads se comunican con otros a través de memoria compartida, y es acá donde se puede presentar el primer problema, al tratar de operar esta memoria compartida entre múltiples threads puede es posible generar bloqueos entre threads.

Para evitar estos bloqueos se usa la exclusión mutua por medio de “lock” programados lo cual permite que solo un thread opere a la vez.
Sin embargo, al realizar exclusión mutua en varias localizaciones, también pueden producir los llamados “deadlocks” en el cual al tener un bloqueo múltiple hace que todos los procesos que dependen de la ejecución inicial se queden totalmente bloqueados hasta que el proceso termine.

Dentro de la implementación en el proyecto, si bien el Mutex se realizó en una sección de código, la cual hace el bloqueo de acceso de memoria de otros procesos, al querer escalar este código a una solución mucho más compleja, seguramente presentaremos problemas de deadlocks.

- Ventajas
    - Implementación sencilla, en la que debemos indicar la porción de código que debemos excluir.
- Desventajas
    - Posibilidad inmediata de presentar deadlocks en la ejecución del programa al momento de excluir el uso de memoria compartida.

STM.

Al trabajar con Software Transactional Memory nos da la posibilidad que suceda o no suceda, asegurando así, que si la transacción es exitosa no tuvo ningún punto de falla, y aunque pueden ser concurrentes ocurren completamente separadas, lo cual es visible directamente cuando usamos Git, un software que implementa este sistema.

Al reemplazar con STM en vez de los bloqueos sobre los threads, nos encontramos con otro tema generado y es que se pueden producir con este sistema operaciones zombies, que nunca harán el commit pero que se siguen ejecutando, por lo que para su implementación se debe asegurar siempre una consistencia interna.

Ahora bien, al realizar esta implementación al proyecto, si bien parecía bastante sencilla que el Mutex, su sencillez se debe al uso interno que se le dio al Mnesia para realizar este versionado, en caso contrario que no se hubiese hecho uso de esta base de datos la implementación hubiese resultado mucho más compleja y engorrosa, llevándolo a utilizar motores de base de datos externos o codificando un motor propio dentro del código que llevara este tipo de versionado de manera consistente.

- Ventajas
    - Nos evita el pensar que se generen deadlocks al realizar las transacciones, esto se logra con el manejo de versiones.
- Desventajas
    - La implementación puede llegar a ser bastante compleja, al tener que generar el tipo de versionado consistente por cada una de las acciones a realizar por cada transacción.
    - La generación de transacciones zombie es uno de los riesgos que puede existir en este modelo.

CSP.

Cuando inicialmente vimos las sentencias **receive do** y **send** en clase, pensaba que el CSP se centraba en este, pero fue una gran sorpresa extrapolar el mismo concepto, hacia los **canales.** El encapsulamiento y sincronización de mensajes nos da la posibilidad de manejar de manera más libre la concurrencia, asegurando la consistencia del envío y recepción de mensajes.

La implementación de este modelo en comparación de las dos anteriores requiere la apertura y cierre del canal, así como el envío y recepción del mensaje, evitando muchos puntos de falla que pueden existir como un mal bloqueo de memoria, o la existencia de un versionado erróneo.

- Ventajas
    - Implementación bastante sencilla
- Desventajas
    - Requiere apertura y cierre de canal.
    - En lo posible es mejor determinar el tamaño del canal

Actors.

Finalmente llegamos al modelo de actores, el cual tuvimos un abreboca bastante llamativo cuando empezamos a ver CSP, pero aprovechando sus principios de trabajo separado y asincronismo la transferencia de mensajes resulta bastante más cómoda que el uso de canales en CSP.

Al elixir ser orientado a actores resultó bastante fácil esta implementación con las sentencias **receive do** y **send**.

- Ventajas
  - Implementación nativa en elixir
  - Bastante más segura ante bloqueos, y seguridad de los mensajes
- Desventajas
  - No encontré desventajas en comparación a los otros tres modelos.

Realizando la implementación de estos cuatro modelos en un mismo proyecto, es visible ver los diferentes cambios que se deben realizar en el código para que este tome forma y funcione correctamente. También es recalcable que Elixir facilita bastante la implementación de estos cuatro modelos, en especial el modelo de actores el cual es nativo.

Hernan David Cuy Salcedo
Código: 202010199