Clase 2

Acciones Atómicas y Sincronización

Atomicidad de grano fino

- **Estado** de un programa concurrente: Valor de todas las variables compartidas y locales, registros, en un cierto instante.
- Cada proceso ejecuta un conjunto de sentencias implementadas con **acciones atómicas**: realizan transformaciones de estado indivisibles, con estados intermedios invisibles para otros procesos.
- Cuando ejecutamos un proceso se resulta en un **intercalado (interleaving)** de las acciones atómicas ejecutadas individualmente.
- **Historia (trace)** son la secuencia de ejecución, algunas no son válidas

Una acción atómica de grano fino se debe implementar por hardware.

- En **sistemas concurrentes**, el **tiempo exacto** no es tan importante como el **orden** de las operaciones.
- Los programas no deben **depender de tiempos específicos**, ya que el **hardware** y el **software** pueden cambiar.
- Lo crucial es que las operaciones sigan una **secuencia lógica**, sin importar el **tiempo absoluto**.
- Los programas deben funcionar correctamente para cualquier **orden** en que las **instrucciones** de diferentes procesos se ejecuten.

Máquinas con acciones de grano fino

- Los **valores básicos** se almacenan en memoria y se manejan como **acciones atómicas**. Esto asegura que las operaciones sean indivisibles, evitando interferencias de otros procesos.
- Los valores se **cargan en registros**, se realizan operaciones sobre ellos, y luego se **almacenan** en memoria. Este proceso optimiza el rendimiento y asegura que los cálculos sean rápidos y seguros.
- Cada proceso tiene su propio conjunto de **registros**, lo que se gestiona mediante **cambio de contexto**. Esto permite que múltiples procesos trabajen de manera independiente y eficiente.
- Los **resultados intermedios** de evaluar expresiones complejas se guardan en registros o en la **memoria privada** del proceso. Esto garantiza que los procesos puedan continuar sin interferencias entre sí.

Expresiones atómicas

- Una **expresión** es **atómica** si no **referencia** una variable modificada por otro proceso, incluso si requiere varias acciones atómicas.

- Una **asignación** (`x = e`) es **atómica** si no **referencia** una variable modificada por otro proceso.
- En programas concurrentes, se deben manejar las **referencias críticas**, que son aquellas a variables que pueden ser **modificadas** por otros procesos.
- Se asume que las **referencias críticas** se hacen a variables simples que se pueden leer y escribir de manera **atómica**.

A lo sumo una vez

1. La **propiedad de "A lo sumo una vez"** asegura que una variable compartida se referencia **una sola vez** en una expresión o asignación.
2. Una **asignación** `x = e` cumple esta propiedad si **e** tiene **una referencia crítica** y **x no es modificada por otro proceso**, o **e** no tiene **ninguna referencia crítica** ninguna (en este caso x si puede ser leída por otro proceso).
3. Si una asignación cumple esta propiedad, su ejecución es **atómica** porque la variable compartida se leerá o escribirá solo una vez.

Especificación de la sincronización

Ejecución atómica: Si una expresión o asignación no cumple con la propiedad **ASV**, a menudo es necesario ejecutarla **atómicamente** para evitar problemas de concurrencia.
Sincronización por exclusión mutua: Para garantizar la correcta ejecución, es necesario que secuencias de sentencias se ejecuten como una única **acción atómica**, evitando interferencias de otros procesos.
Construcción de acciones atómicas: Se utilizan **acciones atómicas de grano fino** para construir una acción atómica de **grano grueso**, que se percibe como **indivisible**.
Especificación de sincronización:

- ⟨e⟩ indica que la **expresión** e debe ser evaluada atómicamente.
- ⟨await (B) S;⟩ especifica la sincronización donde B es una condición de **demora** y S es una secuencia de **sentencias** que termina garantizando que B sea true al inicio de S. Además, el **estado interno** de S no es visible para otros procesos durante su ejecución.

Await

El uso general de `await` (para exclusión mutua y sincronización por condición) tiene un alto costo de implementación. Para `await` en exclusión mutua, se realizan acciones atómicas incondicionales, como ⟨`x = x + 1; y = y + 1`⟩. Para sincronización por condición, se usa ⟨`await (count > 0)`⟩. Si una condición B cumple con ASV, se puede implementar mediante espera activa o spin loop, como `do (not B) → skip od (while (not B); )`.

Propiedades y fairness

Propiedades de seguridad y vida

- **Propiedad**: Atributo verdadero en cada historia de ejecución de un programa.
- **Propiedades de seguridad**: Aseguró que **nada malo le ocurre al proceso** y los **estados son consistentes**, por ejemplo **exclusión mutua**, **ausencia de interferencia entre procesos**, **partial correctness** (corrección parcial, cada vez que termina un programa la ejecución es correcta, si termina).
- **De vida**: Que eventualmente **algo anda**, **progresa** y **no hay deadlocks**. **Falla de vida** indica que **se deja de ejecutar**, por ejemplo **terminación**, asegurar que un **pedido de servicio es atendido**, que un **mensaje llegue a destino**, que un **proceso eventualmente alcanzará su SC**, etc. => dependen de las **políticas de scheduling**.

Fairness y políticas de scheduling

Fairness: Asegura que todos los procesos tengan la oportunidad de avanzar.

Una acción atómica en un proceso es **elegible** si es la siguiente a ejecutarse. Con varios procesos, varias acciones pueden ser elegibles. La **política de scheduling** decide cuál ejecutar.

Ejemplo: Si un proceso ocupa el procesador hasta terminar, podría quedarse en un bucle infinito y nunca terminar.

```
bool continue = true;
co while (continue); // continue = false; oc
```

Fairness Incondicional: Una política de scheduling es **incondicionalmente fair** si cualquier acción atómica elegible **eventualmente se ejecuta**. Por ejemplo, Round Robin (RR) es incondicionalmente fair en sistemas de un solo procesador, y la ejecución paralela en sistemas multiprocesador.

Fairness Débil: Una política de scheduling es **débilmente fair** si:

1. Es incondicionalmente fair.
2. Toda acción atómica condicional que se vuelve elegible **eventualmente se ejecuta**, siempre y cuando su condición sea **true** y permanezca **true** hasta que el proceso vea la acción condicional.

Esto no garantiza que cualquier sentencia `await` elegible se ejecute, ya que la condición podría cambiar (de false a true y de vuelta a false) mientras el proceso está esperando.

Fairness Fuerte: Una política de scheduling es **fuertemente fair** si:

- Es **incondicionalmente fair**.
- Toda acción atómica condicional que se vuelve elegible **eventualmente se ejecuta**, siempre y cuando su guarda se convierta en **true** con **frecuencia infinita**.

```
bool continue = true, try = false;

co while (continue) { try = true; try = false; }

// await (try) continue = false

oc
```

Este programa **no termina** porque la condición `try` se alterna entre `true` y `false` constantemente.

Tener una política **práctica y fuertemente fair** es complejo. En el ejemplo, una política que alterna las acciones de los procesos sería fuertemente fair pero impráctica, mientras que Round Robin es práctica pero no lo es.