Clase 2

Un programa concurrente está formado por un conjunto de programas secuenciales. La programación secuencial puede expresarse con 3 clases de instrucciones

1. Asignación
2. Alternativa (decisión)
3. Iteración (repetición con condición)

Pero para representar esto concurrentemente deberán usarse más clases de instrucciones.

- swap v1:=:v2
- skip: termina inmediatamente y no tiene efecto sobre ninguna variable del programa.
- Sentencia co:

  `co S1// ….// Sn oc:` estructura que permite ejecutar varias tareas concurrentemente. Cada tarea se especifica como S1, S2, S3, … Sn. Se puede pensar en cada Sn como una secuencia de instrucciones que representa una tarea específica. La ejecución de co terminará cuando todas las tareas terminen. Significa que el programa que contiene esta construcción no continúa su ejecución hasta que todas las tareas especificadas de S1 a Sn hayan finalizado su ejecución.

  Cuantificadores: `[i=1 to n]` en `co [i=1 to n] { a[i]=0; b[i]=0 } oc` indica que se crearán n tareas concurrentes, numeradas desde 1 hasta n. En cada tarea se ejecutarán las instrucciones entre {} y cada tarea tendrá un valor diferente para i.

  `Cuantificador: construcción usada para repetir o iterar sobre un conjunto de elementos`
- Process: la palabra clase `process` se usa para crear una representación de concurrencia en la programación. Un proceso se refiere a una unidad de ejecución independiente que puede realizar tareas de manera concurrente con otros procesos.

process	co
Ejecutado en background.	El código que contiene un co espera a que el proceso creado por la sentencia co termine antes de ejecutar la siguiente sentencia.

Acciones atómicas y sincronización

- Estado de un programa concurrente: representación de un momento dado de todas las variables, recursos y estructuras usadas por ese programa en un sistema concurrente.
- Acción atómica: operación indivisible que se ejecuta en su totalidad sin ser interrumpida por otras operaciones concurrentes. Se considera como una sola unidad y que no puede ser dividida o intercalada por otras acciones concurrentes. Algunas operaciones atómicas pueden ser
  - Operaciones de lectura y escritura en variables primitivas.
  - Operaciones de bloqueo y desbloqueo.
  - Inserción y eliminación de elementos en una cola o lista.
  - Operaciones de incremento y decremento atómico.

La ejecución de un programa concurrente es el intercalado (o interleading) de acciones atómicas ejecutadas por procesos individuales.

- Historia (trace): ejecución de un programa concurrente con un interleading particular. En general el número de posibles historias es enorme pero NO todas válidas.
- Interacción: determina cuáles historias son correctas.

La sincronización por condición permitirá restringir las historias de un programa concurrente para asegurar el orden temporal necesario.

Acción atómica de grano fino (fine-grained)

Al hablar de grano fino en contexto de acciones atómicas, se refiere a que las operaciones individuales se realizan de manera muy detallada y específica. Son aquellas que se ejecutan paso a paso y no se pueden dividir aún más. En muchos casos, estas operaciones atómicas de grano fino se implementan a nivel de hardware, lo que significa que el hardware del procesador garantiza la atomicidad de la operación.

```
(i) Load PosMemB, reg
(ii) Store reg, PosMemA
```

Estos dos pasos separados no son atómicos en el sentido de que podrían ser intercalados por otras operaciones concurrentes. Esto significa que otro hilo o proceso podría leer `B` después de que se haya cargado pero antes de que se almacene en `A`, lo que podría llevar a problemas de concurrencia.

```
X= X+X
(i) Load PosMemX, Acumulador
(ii) Add PosMemX, Acumulador
(iii) Store Acumulador, PosMemX
```

Tampoco es atómica porque hay varios pasos. En conclusión para lograr atomicidad en operaciones no atómicas como las mencionadas, generalmente se requiere el uso de mecanismos de sincronización, como operaciones atómicas proporcionadas por el hardware. Hay que tener en cuenta en la atomicidad de grano fino:

- El tiempo absoluto no es importante. El sistema se centra en la secuencia de eventos y en cómo se relacionan entre sí en lugar de cuándo ocurren exactamente.
- Actualizaciones de componentes más rápidas: la corrección del sistema no debe depender de tiempos absolutos predecibles, ya que estos pueden cambiar con las actualizaciones de hardware o la carga de trabajo.
- El tiempo se ignora, solo las secuencias son importantes: en lugar de preocuparse por el tiempo absoluto, los sistemas concurrentes tienden a centrarse en la secuencia de eventos. Importa cómo se relacionan las operaciones entre sí, el orden que adquieren o liberar recursos.
- Diferentes órdenes de ejecución (interleavings): puede haber múltiples órdenes en los que se ejecutan las instrucciones de diferentes procesos debido a la naturaleza de la concurrencia.

A tener en cuenta

- Si una expresión `e` en un proceso no hace referencia a ninguna variable que sea modificada por otros procesos, la evaluación de `e` se considerará atómica dentro de ese proceso, incluso si requiere la ejecución de múltiples acciones atómicas de grano fino internamente. Esto significa que la evaluación de `e` se llevará a cabo sin ser interrumpida por otros procesos concurrentes y se

garantiza su integridad, ya que no depende de acciones concurrentes que modifiquen las variables a las que hace referencia.

Si una asignación x = e en un proceso no referencia ninguna variable alterada por otro proceso, la ejecución de la asignación será atómica.

- Referencia crítica: referencia a una variable que es modificada por otro proceso. Asumamos que toda referencia crítica es a una variable simple leída y escrita atómicamente.
- **A lo sumo una vez:** una sentencia de asignación `x= e` satisface la propiedad *a lo sumo una vez* si

  1. e contiene a lo sumo una referencia crítica y x no es referenciada por otro proceso, o
  2. e no contiene referencias críticas, en cuyo caso x puede ser leída por otro proceso.

  Una expresión e que no está en una sentencia de asignación satisface la propiedad *a lo sumo una vez* si no contiene más de una referencia crítica.

Si una sentencia de asignación cumple la propiedad ASV, entonces su ejecución parece atómica, pues la variable compartida será leída o escrita sólo una vez.

Especificación de la sincronización

Si una expresión o asignación no satisface ASV con frecuencia es necesario ejecutarla atómicamente. Es necesario ejecutar secuencias de sentencias como una única acción atómica (sincronización por exclusión mutua).

Hay un mecanismo de sincronización para construir una acción atómica de grano grueso como secuencia de acciones atómicas de grano fino que aparecen como indivisibles

- `<e>` indica que la expresión e debe ser evaluada atómicamente
- `<await (B) S;>` se usa para especificar sincronización. La expresión booleana B especifica una condición de demora. S es una secuencia de sentencias que se garantiza que termina. Se garantiza que B es true cuando comienza la ejecución de S. Ningún estado interno de S es visible para los otros procesos.

`await` es una sentencia de alto poder expresivo pero el costo de implementación es alto.

- Await general: `<await (s>0) s=s-1;>`
- Await para exclusión mutua: `<x= x+1; y=y+1>`

Ejemplo await para sincronización por condición `<await (count>0)>`. Si B satisface ASV, puede implementarse como busy waiting o spin loop.

Propiedades y fairness

Las propiedades de un programa concurrente son características deseadas en cualquier posible historia. En otras palabras, es una declaración sobre cómo debería comportarse el programa en cualquier historia.

Toda propiedad puede ser formulada en términos de 2 clases

- Seguridad (safety): se centran en garantizar que nada malo ocurra en un programa concurrente. Asegurarán que el sistema esté en un estado consistente y que se eviten problemas graves. Algunas propiedades de seguridad son

    - Exclusión mutua: garantiza que dos o más procesos no puedan acceder simultáneamente a un recurso compartido.
    - Ausencia de interferencia entre procesos: asegura que los procesos no interfieran entre sí y que sus acciones no tengan efectos no deseados en otros procesos.
    - Partial correctness: garantiza que un programa cumple con su especificación y que si termina lo haga correctamente.
- Vida (liveness): estas propiedades se refieren a que eventualmente “algo ocurre algo bueno”en la ejecución de un programa concurrente. Se centran en el progreso y la prevención de situaciones en las que el sistema se bloquea o deja de funcionar.
    - Terminación: asegura que un programa eventualmente termine su ejecución y no quede atrapado en un bucle infinito o deadlock.

Fairness y políticas de scheduling

- Fairness: trata de garantizar que los procesos tengan chance de avanzar sin importar qué hagan los demás.

Una acción atómica en un proceso es elegible si es la próxima acción atómica que será ejecutada. *Si hay varios procesos hay varias acciones atómicas elegibles.*

- Política de schedulling: determina cuál será la próxima en ejecutarse.
- Fairness incondicional: política de scheduling en el que se garantiza que cualquier acción atómica que sea elegible para ejecución eventualmente se ejecutará. Significa que si un proceso está listo para ejecutarse y no está bloqueado, el sistema de scheduling se asegurará de que ese proceso tenga la oportunidad de ejecutarse en algún momento.
- Fairness débil: una política de scheduling se considera "débilmente justa" si cumple con dos condiciones fundamentales: es incondicionalmente justa, lo que garantiza que las acciones atómicas incondicionales sean tratadas con justicia, y también garantiza que las acciones atómicas condicionales se ejecuten si su condición se convierte en verdadera, siempre y cuando la condición permanezca verdadera hasta que sea vista por el proceso que ejecuta la acción. Esto asegura que todas las acciones, independientemente de su condición, se gestionen de manera justa y que el sistema haga un progreso adecuado.

No es suficiente para asegurar que cualquier sentencia `await` elegible eventualmente se ejecuta: la guarda podría cambiar el valor mientras un proceso está demorado.

- Fairness fuerte: una política es fuertemente fair si
    - Es incondicionalmente fair y
    - Toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada pues su guarda se convierte en true con infinita frecuencia.