Clase 2

lunes, 19 de agosto de 2024 10:30

Acción atómica: acción que nadie pede interrumpir mientras hago. Nadie puede ver variable que modifico hasta que termino.

Cada proceso tiene sentencias para ir ejecutando. Se van intercalando acciones atómicas de los distintos procesos
Historia: cada ejecución de un programa. En general el número de posibles historias de un programa concurrente es enorme; pero no todas son válidas (porque está mal programado).

Interacción: determina qué historias son correctas.

Hay que asegurar orden temporal para no cagarla.

Sincronización por condición: restingir historias para asegurar el orden temporal

Acción atómica de grano fino: De máquina, se implementa por hardware. != las que nosotros codeamos. Esas no son atómicas pero necesitamos que lo sean .
¿La operación de asignación A=B es atómica?
Þ (i) Load PosMemB, reg
(ii) Store reg, PosMemA
A=B no es atómica, pero el load y store lo son (de grado fino).

```
x = 0; y = 4; z=2;
co
  x = y + z        (1)
  // y = 3         (2)
  // z = 4         (3)
oc
```

(1) Puede descomponerse por ejemplo en:
(1.1) Load PosMemY, Acumulador
(1.2) Add PosMemZ, Acumulador
(1.3) Store Acumulador, PosMemX
(2) Se transforma en: Store 3, PosMemY
(3) Se transforma en: Store 4, PosMemZ

Cuando uso co-oc cada proeso se separa con las //
Todo lo de adentro de la 1 se ejecuta en ese orden si o si, pero puede ser que por ejemplo 2 y 3 se ejecuten entre ellas

```
x = 2; y = 2;
co
  z = x + y        (1)
  // x = 3; y = 4; (2)
oc
```

(1) Puede descomponerse por ejemplo en:
(1.1) Load PosMemX, Acumulador
(1.2) Add PosMemY, Acumulador
(1.3) Store Acumulador, PosMemZ
(2) Se transforma en:
(2.1) Store 3, PosMemX
(2.2) Store 4, PosMemY

NO es 6 un valor de estado válido pq eso implicaria que quedó el y con 4 nuevo y el x viejo:2. Incoherente

Tiempo:

No puedo asumir cuánto tiempo va a tardar en tiempo absoluto. No puedo asumir que no se van a pisa, tengo que sincronizarlos para que no se pisen.

- Leer y escribir son atómicas de grado fino
- Valores se cargan, se opera y se almacenan.
- Cada proceso tiene sus registros propios (cuando hago context switch no se pierde la data)
- Si todo lo que hago es con variables de mi proceso, aunque sean varias acciones de grano fino no hay drama, no tendria problemas

- SI una asinación no referencia nada que altera otro proceso, es grado fino.

Referencia crítica: referencia a una variable modificada por otro proceso: modificar un dato que pueda estar siendo modificada por otro proceso.

> Una sentencia de asignación $x = e$ satisface la propiedad de ***"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 expresiones e que no está en una sentencia de asignación satisface la propiedad de ***"A lo sumo una vez"*** si no contiene más de una referencia crítica.

Propiedad de a lo sumo una vez:
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.

- Si estoy trabajando con variables compartidas pero solo estamos leyendo, no hay drama.
- A lo sumo puedo estar referenciando una variable compartida una vez. Tener una sola referencia crítica.

Si la que es crítica es x, e no puede ser modificada por otra?
Se evala sentencia a sentencia.
SI esa sentencia hace referenci una vez a una sola variable que sea jodida?

Si sentencia cumple ASV, la puedo ejectar como si fuera atómica
No importa si me puede dar valores distintos. El tema es cuando por ej una se modifica y una no. Ejemplo: si yo paso plata no pasa nada si da raro. Está mal si los valores de la trasferencia no tiene sentido. Pierdo/gano plata.

Ejemplos:

- int x=0, y=0;
  co x=x+1 // y=y+1 oc;
  No hay ref. críticas en ningún proceso.
  En todas las historias $x = 1$ e $y = 1$

- int x = 0, y = 0;
  co x=y+1 // y=y+1 oc;
  El 1er proceso tiene 1 ref. crítica. El 2do ninguna.
  Siempre $y = 1$ y $x = 1$ o 2

- int x = 0, y = 0;
  co x=y+1 // y=x+1 oc;
  Ninguna asignación satisface ASV.
  Posibles resultados: $x = 1$ e $y = 2$ / $x = 2$ e $y = 1$
  Nunca debería ocurrir $x = 1$ e $y = 1$ → ***ERROR***

Necesito sincronizar las acciones que no son de grado fino ni se comportan como tal.
Sincronizazción por exclusión mutua: decir que este fragment de código se tieene que ejecutar sin que nadie modifique en el medio. Armar acciones de grano grueso.

- AWAIT: (await(b)S;) se espera hasta que se cumpla el boolean B. S es lo que se ejecuta como si fuera de grano fino.
  Los estados internos de s son invisible para el resto de procesos
  Atómica desde que se cumple b hasta que termina. Una vez terminado se libera la atomicidad.
- (e) indica que la expresión e debe ser evaluada atómicamente. No tiene condición.
- (await (count > 0)) sincronización por condición
  - Si B cumple con el a lo sumo una vez, puedo reemplazarlo por (while (not B);). Es un busy waiting, demora un proceso hasta que no se de una condición. No es eiciente pq usa proce. Es mejor sacar al proceso, que otro modifique la condición y después ponerlo de nuevo. Que no tenga el proce al pedo.

Poner los <> no me asegura que no me lo van a tocar. Para asegurarme no cagarla todos los lugares criticos tienen que usarlo

No se puede interactuar con un buffer en simultáneo (cola, pila, loquesea) solo podria si es un arreglo y estoy segura de que están en distintas posiciones.

<push> y <pop>, pero tambiÍen los aumentos y decrementos --><push, cant+1> y <pop, cant-1>

```
cant: int = 0;
Buffer: cola;

process Productor
 { while (true)
      Generar Elemento
      <await  (cant < N); push(buffer, elemento); cant++ >
 }

process Consumidor
 { while (true)
      <await  (cant > 0); pop(buffer, elemento); cant-- >
      Consumir Elemento
 }
```

Si yo no hago el cant <n y lo hago con un if, se lo salta. No me sirve

Propiedades y fairness

Propeidad: atributo verdadero en cualquier historia

- Seguridad
  - Nada malo va a pasasr, estados consistentes.
  - Corrección parcial: si un programa terminó, el programa tiene un vlaor correcto (puede no terminar)
- Vida
  - Nos asegur que procesos van avanzando
  - Terminación: seguro en cualquier ejecución que haga va a terminar(no sé si bien o mal pero no se cuelga)
  - Aegurar ausencia de deadlock

Vida-> termina
Seguridad-> termina bien

Fairness: tratar de que todos tengan chance de avanzar
Pollitica de scheduling: para ver las instrucciones de qué proceso voy a ejecutar. La idea es que todos puedan ir avanzando.
Acción atómica es elegible si es a próxima a ejecutar de ese proceso en particular.
La acción atómica elegida puede ser condicional o incondicional

Fairness incondicional: si toda acción atómica incondicional elegible es eventualmente ejecutada. No nos ejecuta nada sobre las condicionales.

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

En el ejemplo anterior, RR es incondicionalmente fair en monoprocesador, y la ejecución paralela lo es en un multiprocesador.
Fairness debil:

- Es incondicionalmente fair
- Toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada, asumiendo que su condición se vuelve true y permanece true hasta que es vista por el proceso que ejecuta la acción atómica condicional.

- Problema si hay una variable que varios procesos cambian. Puede ser que continuamente esté false true false y que cada vez que se analiza está ocupada
- Round robin cumple

Fairness fuerte:

- Es incondicionalmente fair
- Toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada pues su guarda se convierte en true con infinita frecuencia.
- Imposible implementar. Vamos a usar las otras dos.

EXPLICACIÓN DE PRÁCTICA:

Ejemplo 2:
Si tengo que usar variable para avisar algo, quien recibe el mensaje es quien lo resetea para la siguiente vuelta