swap: v1 :=: v2 → El valor que estaba en v1 queda en v2 y al revés(no recomendable usarlo)

**skip** no produce cambio, se utilizar para completar las estructuras de control

Evalúa todas las condiciones booleanas y donde es verdadera elige una de forma aleatoria y elije el conjunto de sentencias asociadas con él.

Si cuando termina todas son falsas sale del if sin hacer nada mas

**iteración**

Partes secuenciales, while, for

for all mas de un cuantificador, i y j, el for itera por todas las combinaciones i y j dadas por los limites.

St → nos inidca uqe para que esa combinacion de i j se pueda completar la ejec debe cumplir para i j , si para esa comb no cumple, entocnes no se ejecuta nada y pasa a la sig.

alternativas iterativ multiple, if mutiples alternativas pero itera sobre el, do en vez del if, cada una func booleana un conju de sentencias.

Diferencia con el if , (ejecuta una y termina)

el otro vuelve a iterar chequea todas las condicioones, y asi itrando hasta uqe encuentra que todas ls cond booleanas son falsas.

si se solapan ls tres por ejempplo, se define de forma deterministica cual es la correcta

for all :

Cuales son los prcesos que se van a poder ejec de forma concurrente

Concu en mem distr y mem compartida.

co → unico proceso principal , ejeuta un unico hila, hasta que neceistamos algo de forma concurrente, entonce ponemos la sentencia **co,** en ese instante se generan una serie de hilos dieferetes donde cada uno va a ejecutar un ppedacito de codigo, cuando terminan esos hilos se juntan en uno, si en otro momento podemos volvemos a llamar a la funcion on y genera distintos hilos de uelta.

Dos maneras de hacer lso llamadados, que va a dependender si son distints o no los pedacitos de los hilos que se ejecutan, si cada uno tiene q hacer algo diferenete, cada pedacito de sentencia de codigo co S1 // ..... // Sn oc delimitador que delimita a done termina cada uno de los procesos. esto para un codigo diferente cada hilo.

si todos lso procesos tuvieran qe hacer el mismo trabajo, podemos definir un cuantificador, un co con i de 1 a n y un codigo asoc . n hilos

co [i=1 to n]{ a[i]=0; b[i]=0 } oc → Crea n tareas concurrentes.

se distinguen lo hilos por el valor de la variable i, el primero 1. el segundo 2, .

asi diferenciamos cada uno de los procesos que ejec de forma concurrente.

**Process**

cada uno define lso distintos procesos q forman parte del prog conc y esos procesos, corren dsde el inicio, después algunos terminan antes y uqedan los uqe tienein codigo, todos arrancan juntos. Todos los hilos tienen la misma prioridad y cominezan juntos.

Esto lo utilizan para pasaje de mensaje. Trabajan con procesos y no con threads.

Process: otra forma de representar concurrencia process A{sentencias} → proceso único independiente. Cuantificadores. process B [i=1 to n] {sentencias} → N procesos independientes.

• Diferencia: procesos ejecuta en background(entrada todo el tiempo), mientras el código que contiene un co espera a que el proceso creado por la sentencia co termine antes de ejecutar la siguiente sentencia.

ejecuta un ihlo y cuando es necesario algo de forma concurrente.

segundo ejemplo paralelizan y tener 10 procesos. y cada uno que imprima el valor del cuantificador, ambos progrmaas son equivalentes? pues no , 1 imprimer todos del 1 al 10, en el segund en cualquier orden.

**Acciones atómicas**

Estado de un programa concurrente

El valor que tienen las variables compartidas de cada proceso en cierto momento.

Si uno corta la ejecución de los proceso y guarda esos valores ese seria el estado de un programa.

cuando un proceso tiene q ejecutar un programa concurrente lo que ejecuta es una serie de sentencias → formadas por acciones atómicas. cada proceso ejecuta una acción atómica cada acción atómica lo que hace es una transformación de estado indivisible, los estados intermedios que se pueden llevar a cabo mientras se está llevando a cabo esa acción, no se ve por los otros , por ende no puede haber una interferencia. .

intercalado de las acciones atomicas pertenecientes a los distintos procesos.

secuencia de estados o acciones o dentencias tom, forman una historia, esa historia no todas pueden ser válidas.

uno tiene que evitar las sentencias que no son buenas.

asegurar el orden necesatro

La sincronización por condición permite restringir las historias de un programa concurrente para asegurar el orden temporal necesario apra resolver el problema.

en algun momento le agrego la sincroniacon uqe nso asegur que se cumlpa la semantica del ejecicio.

Una acción atómica de grano fino (fine grained) se debe implementar por hardware.

ya que son atómicas no debe generar nada por sw para armarla, ni poner sincro, de estar arma las de grano grueso, por sw, que debo sincronizarlas las sentencias por exclusión mutua para que se de la atomicidad que necesita.

Implementadas por hw, sentencia de maquina, una op de asginacion a:= b, no la consideramso atomica porq internamente involucra dos acciones cargar el dato de la pos de meme b en un reg del proc y otra donde guardo el valor de ese reg en la dir de mem dado por la varaible a, dos acciones atómicas,

x := x + x

tres atomicas de grado fino

carga el valor en un reg del procesador, segunda sentencia a ese sumarle el valor de la segunda x, y luego al resultado de la suma en el AC guardarlo en la posi de mem de X.

(i) Load PosMemX,Acumulador

(ii) Add PosMemX,Acumulador

(iii) StoreAcumulador, PosMemX

x:=y+z → 3 acciones atomicas de grano fino

siempre x y z el mismo valor y x puede variar por un no determinismo.

vemos que si queremos intercalar las acciones atomicas nos pueden dar los resutlado del ejemplo.

1.1 y cargamos el valor de la varaible Y en un reg.

valores posibles por no determinismo.

Un inconveniente qeu ssi podria darnos un valor incorrecto, dos procesos, uno la suma y otro las modificaciones,

x = 3, y = 4 en todos los casos. z puede ser: 4, 5, 6 o 7. Nunca podría parar el programa y ver un estado en que x+y = 6, a pesar de que z = x + y si puede tomar ese valor

no existe la situacino en que x + y sume 6, pero es un posiible resultado de la varaible z

En la mayoría de los sistemas el tiempo absoluto no es importante.

Con frecuencia los sistemas son actualizados con componentes más

rápidas. La corrección no debe depender del tiempo absoluto.

El tiempo se ignora, sólo las secuencias son importantes

Puede haber distintos ordenes (interleavings) en que se ejecutan las instrucciones de los diferentes procesos; los programas deben ser correctos para todos ellos.

poner toda la sinc necesaria para que una snetencia que sea posterior a otra se haga asi, la tenemos que hacer por software.

En lo que sigue, supondremos máquinas con las siguientes características:

• Los valores de los tipos básicos se almacenan en elementos de memoria leídos y escritos como acciones atómicas. Si tengo que leer o escribir, una entera simples van a ser acciones atom de grano fino.

• Los valores se cargan en registros, se opera sobre ellos, y luego se almacenan los resultados en memoria. Como en los ejemplos anteriores en la suma.

• Cada proceso tiene su propio conjunto de registros (context switching). Cada vez que deje el procesador guardar su ed para guardar los valores y desp vuelve y los tiene.

• Todo resultado intermedio de evaluar una expresión compleja se almacena en registros o en memoria privada del proceso. No hay forma de que algo intermedio sea guardado en una variable compartida, lo que se ejecuta sentencia no va a cambiar ciertos valores interiores.

Distintos proc ejec sus sentencias o grupo de sentencias de forma atómica, mientras se esta ejecutando algun grupo de sentencia que utilice o no variable compartida ningún otro pueda interferir y modificar variables o datos con las que trabajaba ese proceso. Si un grupo de sentencias que no tiene variables compartidas no importa cuantas sentencias atom de grano fino se incluyan dentro de ella se va a ejecutar atómicamente. En cambio, a variabls comp pero que no son modif por otros procesos, aunque no parezca qeu sea atom porque puede tener varias secciones atom de grano fino, igua se va a estar ejectuando de esa forma porque nadie puede modif su resultado.

Alguns casos en lo que si son modif por otros procesos en el transcuros de su ejecucuion, ¿como lo resuelvo?

requerim mas debil,

ref critica

Referencia crítica en una expresión ⇒ 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 una variable por otro proceso, en el ejempli a una variable simple.

Propiedad de a lo sumo una vez, va a ejec como si fuese atomica, aunqe no sea atom y aunque haga ref a variables compartidas y modif por procesos,

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 → si en e se hace referencia a una variable compartida que es modificada por otro proceso a lo sumo una, entonces nadie puede estar accediendo a X porque podemos tener un problema de interferencia.

2) e no contiene referencias críticas, en cuyo caso x puede ser leída por

otro proceso. → e solo hace ref a variables locales, en ese caso x si puede ser leída por otro proceso que no hay inconveniente.

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. → a lo sumo ref a una var compartida y a lo sumo una sola variable compartida.

Ejecucion de una expresión, condicion en if, satisface la prop si no contiene mas de una ref critica, una vari compartida por otro proceso a lo sumo una vez.

Su ejec parece atómica aunque esto involucre una serie de acc atomica de grano fino y a su vez a una var compartida por otro proceso, porque la var compartida a lo sumo una vez.

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

int x=0, y=0; No hay ref. críticas en ningún proceso.

co x=x+1 // y=y+1 oc; En todas las historias x = 1 e y = 1

int x = 0, y = 0; El 1er proceso tiene 1 ref. crítica. El 2do ninguna.

co x=y+1 // y=y+1 oc; Siempre y = 1 y x = 1 o 2

int x = 0, y = 0; Ninguna asignación satisface ASV.

co x=y+1 // y=x+1 oc; Posibles resultados: x= 1 e y =2 / x = 2 e y = 1

Nunca debería ocurrir x = 1 e y = 1 → ERROR

*si cumple con la prop se puede ejecutar como si fuera atómica*

puede haber errores si no ocurre la pro ASV → entonces necesito una herramienta, algo que me permita decidir qué tengo que ejecutar un conjunto o más, que necesito si o si, ejecutarla de forma atómica como si fuese una acc atom aunque no cumpla ASV, aunque haga referencia a muchas variables compartidas mdif por otros proc o aunque incluya muchas sentencias que cumplan alguna relación entre ellas y nadie debería poder interferir en el medio sin que ellas terminen de ejecutar, es lo qeu se llama  **sincronización por exclusión** mutua , programar un bloque que esté seguro que se van a ejecutar sin que ningún otro proc en el medio pueda interferir.

Necesitamos Mecanismo de sincronización para construir una acción atómica de grano

grueso (coarse grained) como secuencia de acciones atómicas de grano fino

(fine grained) que aparecen como indivisibles.

〈await (B) S;〉 se utiliza para especificar sincronización.

La expresión booleanaBespecifica 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.

esta sentencia nos permiten ejecutar un conjunto de sentencias de forma atómica para desarrollar soluciones con pseudo codigo, desp vemos como simulamos este tipo de sentencias. También para sinc por condicion.

Entre menor y mayor, la sentencia o conj de sentencia que tenemos q ejec como una acc atom de grano grueso, y no s aseguramos que ningún otro pueda ejec ese pedacito de codigo, si dentro de la expresión hacemo referencia a la variable compartida a y hay otro proc modif esa variable sin haber definido una acción atom de grano grueso, igualmente podría , la expreion la unica manera es que ambso rpocesos cuando hagan ref a una variable compartida es que lo hagan dentro de esta sentencia.las qeu vimos son sentencias atomicas incondicionales, una vez que logra acceder de forma exclusiva a ese bloqeu de sentencias entra a ejec, sino espera hasta q este libre entra y lo ejecuta. se llama incondicionales pero las await t nos permiten condicionales, refiere unir sin por **exclusion** mutua con la de **condicion**, el proceso con exclusión mutua va a tener que esperar hasta que la condicion b sea verdadera y cuando b es verdadera va a pdoer ejec el conj de sentencias sin salir de condición mutua, si es falsa espera, si es ferdadera sin salir de esta exlusion ejec las sentencas, mientras chequeo qeu la condici es verdadera nadie en el medio haya podido modif el valor de esa condición, espera hasta q la cond B sea verdadera por condición y una vez que es erdades sin salir ejecuta , ahi porp exclusión mutua, esta sentenci a su vez, podría no requerir ejecutar un conj de secuencias sino que

esta sentencia que es la completa, espera hasta que B sea verdadera, ahi nos da la sinc por cond, y una vez q es verdadera ejecuta e ocnj de sentencias, sinc por exclusión mutua.

sta sentencia a su vez podría no requerir el conjunto, sino que solo por condición sin S

costo muy alto implementar esto, soluciones con pseudo código, el await general es el qeu dice q el proceso par poder ejec esa acción atomica tiene q esperar que la variable S qe es compartida sea mayor que 0, y cuando encuentra uqe es mayor que cero ejec de forma atom, si es =0 , libera la sección esta y se queda esperando apra poder volver a entrar, la libera asi otro puede ingresar y asi modificar el valor y llegar a lo esperado.

• Await general: 〈await (s>0) s=s-1;〉

si encuentra qeu s es igual a 0, sale de esta seccion y se queda esperando para poder voler a entrar, cuando entra porqeu esta libre el uso de s y chequea la condicion si la encuentra mayor que 0, otomaicamente decrementa en uno esa varaible, en el segundo cuadro vemos un ejem cuando la sentencia se utiliza unicamente por exclusion mutua, atomicamente se tiene que decrementar y en 1 e x en 1.

el proceso va a esperar ahsta que la var compartida count sea mayor qeu 0.

esperar hasta que una condicion booleana es verdadera. en este caso si esa expresion booleana cumple con la prop, a lo sumo hace ref una var ocmpartida ASV, esa setencia se peude implementar

Ejemplo await para sincronización por condición: 〈await (count > 0) 〉 Si B satisface ASV, puede implementarse como busywaiting o spin loop do (not B) → skip od (while (not B); )

busy waiting, esperar una estrucit de control iterativa, espera hasta q esa cond sea verdadera.

while not b, me quedo iterando cuando salimoms esp orque la condcion es verdadera. En base a esta tecnica de busy waiting a implemetar a través de un wait es lo qeu nos va a permitir empezar a trabjar con el uso de variables compartidas en codigo verdadero no pseudo codigo.

cuando la sentencia tiene que usar la condicon lleva await y la condicion, sentencias atomicas condicionales.

se bloquea para poder continuas la condicion, necesita que la condicion sea verdadera, sino bloquea hasta q sea verdad.

ejemplo del buffer, ambos tipos, de condicion para que no deje un buffer lleno ni lo descarte y tmp qe un consumidor tome un elemn de un buffer vacion ni que siga haciendo otra cosa, ademas de generar un element desp de generado esperar hasta que la cantidad de elem sea < N , igual a N espero , si es meor que N sin alir del uso exclusiivo deposito en el buffer y desp increm los elem

atomicamente deberia sacar los elem del buffer.

cuadno accedeemos a una cola, no se puede al msmo tiempo estar haciendo dos procesos un push , o pop, uno solo modificando el valor del buffer sea tomando o sacando cosas de el

**Clase 6.**

**Propiedades y Fairness**

propiedad: una caract que es verdadera en cualquierda de las posibles historias que puedan darse en un pprogrma concurrente, esa caracte debe ser verdadera en cualquier caso, esas prop puede clasif en dos tipos:

De **seguridad** , es una caract que nos asegura que nada malo le ocurre a un proceso, todos los estados son consistentes, si algo funciona mal error en el prog, por ejmplo: si tenemos que usar cierto recurso compartido de forma exclusiva y puede en algun momento dos procesos estar usando ee recurso compartido es un error uqe diece q algo está funcionndo mal y no se ucmpliria una prop de seguridad. Ejemplos de propiedades de seguridad: exclusión mutua, ausencia de interferencia entre procesos, partial correctness → correccion parcial siemrpe que un programa termina su ejec la finalizaicioon es correcta lo que no nso asegura que siempre termine, nos asegura que nada malo ocurre en el medio en cuanto a la forma en que se desarrolla el programa.

de **vida**: los procesos van a poder continuar en algun momento con su ejecución, para que el proceso progrese en su ejecucuón, uno o mas procesos no peudan continuar su trabajo una falla de vida.

Ejemplos de vida: terminación, asegurar que un pedido de servicio será atendido, que un mensaje llega a destino, que un proceso eventualmente alcanzará su SC, etc ⇒ dependen de las políticas de scheduling.

¿Que pasa con la total correctness?¿Que pasa con la total correctness?

aquella en la cual nos asgura, la correccion entre la parcial y la terminacio, nos asegura como la terminacion que el prgo siemrpe terminara pero tb nso asegura que termina bien, union de ambas propiedades.

garantizar qeu todos los rocesos puedan avanzar y cont con su ejec, haciendo algo que requieran hacer, puedan avanzar e n su ejec. una accio natomica en un proceso particular es elegible si es la proxima accion atomica que debe ejecutarse en ese proceso, mas alla de qeu pueda ejecutarse o no, hay que ver cual se puede ejectuar en ese mometno.

scehduler determina cual es la proxima a ejecutarse en ese progrmama concurrente para formar parte de la historia qeu se está armando

ejemplo: unica unidad de proc so que determ cual es el proc que puede cont su ejec, la política de scheduling dice se le da el procesador a una unidad de procesamiento y hasta uqe no termina o se demora por motus propio no puede sacarse del uso del procesador,

y que pasaria en el ejemplo, dos proc concurrentes,

bool continue = true;

co while (continue); // continue = false; oc

que ocurre si el proceso uno gana el procesador, si gana el programa nunca terminara para poder dejar el procesador es terminar de ejecutar el proceso o que ejecute alguna sentencia de demora, quedaría en un loop. CC el proceso uno nunca ejecutaría nada porque el valor en falso entonces el proceso uno ni entra al while una vez, sale de la primer iteración y termina.

las políticas de scheduling se clasifican de tres maneras a medida de que trabajan con distintos tipos de sentencias.

**Fairness Incondicional.** Una política de scheduling es incondicionalmente(no esta con await o estaba con await pero sin la condición booleana solo para exclusion mutua) fair si toda acción atómica incondicional que es elegible eventualmente es ejecutada. En el ejemplo anterior, RR es incondicionalmente fair en monoprocesador, y la ejecución paralela lo es en un multiprocesador.

Cualquiera de las acciones elegibles, incondicional en algún momento se va a ejecutar, se le va a dar el permiso de continuar y como no tiene condición, se va a ejecutar si o si,cuando se leda permiso paara ejecutar a ese proceso en particular,es uno el round robin .

ese tipo de sentencias seguro van a poder avanzar, no nos asrgura con las qe son condicionales, el wait con la condición booleana, n onos asegura que en algun moemtno se ejecuta, no vamos a estar seguros puede que nucna pase ee punto, no se cumple prop de vida, no es uqe nunca lo vyaa a hacer pero no estamos seguros, entonces surgen las

**Fairness Débil.** Una política de scheduling es débilmente fair si : (1) Es incondicionalmente fair y (2) 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.

una vez que se volio verdadera no vuelve a flaso hasta qeu podamos ejecutarla, condición x verdader y el unco que puede vovlerla a falo es el tipo de cond atomcia, si es debilmente fair nos aegrua que se puede ejecutar n algun mometo, si alguien la puso verdadera como somso los unico uqe podemos pasar a falsoentocnes el proceso va a pdoer continuar con su trabajo.

condicionales donde verdadero y falo peuda ir alternando, un oen verdadero otro en falso, sin que uno que está esperando ponerla en verdadero y pueda ejecutarla.

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

(1) Es incondicionalmente fair y

(2) Toda acción atómica condicional que se vuelve elegible eventualmente es

ejecutada pues su guarda se convierte en true con infinita frecuencia.

bool continue = true, try = false;

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

// 〈await (try) continue = false〉

oc

podria ser que cada vez que le proceso dos intenta ejec esta senennncia wait encuentra uqe la sentencia await encuentra falso, podria ser que el procesos uno termina su quantum en el moemotn uqe pasa a falso, pone en falso, lo pasa al otro proceso, nuevamente alcanza a ejec una itreacion completa pero la vuelve a poner en falso, asi cada vez encuentra falso entonces no puede ocntinuar, round robin débilmente fair ,entocnes entra la politica doblemene fair, si su condicion booleana en algun moemnto la pone en true seguro que en algun momeont se encuentra verdader y cont su ejec.

No es simple tener una política que sea práctica y fuertemente fair. En el

ejemplo anterior, con 1 procesador, una política que alterna las acciones de

los procesos sería fuertemente fair, pero es impráctica. Round-robin es

práctica pero no es fuertemente fair.

clase 17/9

no suponer que un proceso como va a hacer mas cosas vaya a llegar mas tarde, no pensar que primero termina al que tenga menos cosas porque pudo haberse bloqueado e intenten modificar la variable en el mismo momento.

no sabemos la máquina ni la velocidad en la que ejecuten.

ASV: que si van a generar interferencia, incrementa la cantidad de sentencias,

con dos o mas sentencias que se tienein qeu ejecutar en algun mometno voya tener problema en ejecutarlas opuedo ejecutarlas sin problemas, si tenemos una expresino

cumple si tienen a lo sumo una referencia critica, una var ref por otro proceso, y a su vez no esta siendo leida pr nadie cumpe la prob, entocnes la puedoejecutar no me da un valor erroneo.

la voy a poder ejecutar sin problema, nno va a generar interferencia, ejecutar lo de exclusion mutuas.

referencia critica querer accedeer

sentencias pr exclusino mutua, hasta ahora no tenemos forma de implementarlo, < forma atomica>

continuar avanzando con su ejecución, fairness implica que todos tengan chance de pasar