Acciones atómicas y como y cuadno se deben sincronizar

LAs herram que nos van a permitir asegurar esa atomicidad

estado es el valor uqe tienine las varibles, compatidas, locales , valor de losregistros, en un instante de tiempo, si uno corta la ejecucion de los rprocesos y guarda un estado ese seria el estado de un programa concurrentee.

Cada sentenciaesta formada por una o mas acciones atomicas, transformación de estados invisibles, acciones ne le medio que se llevan a cabo no son visibles para los demas procesos, la secuena de estado o acciones o sentenncias atomicas, formsn una historia, no otdas esas hsitorias son válidas → la interaccion entre procesos nos permiten dif cuales sera validas y cuales no.

que se cumpla el orden teemporal necesiario parauqe las hist sean validas, porr cond nos va a permitir las historias posibles, para resolver el problema que tenemos que implementar.

cuales son acciones atom de grano fino? ya atomicas, de grano fino, no debe generar nada de sw paa armarla, en base a estas arma las de grano grueso se arma por sw, sincronizar sentencias por xlicsion mutua para quese de esa atomicidad pero no existe.

ggrano fino de maquina

a=B

no la vamos considerar atómica, involucrado acciones, cargar los datos que esta e ka pos de mem b en un reg del procesador y otra donde guarda el valor del reg del procesador. (dos acciones atómicas) si se ejecuta podria ser q entre la sentencia 1 y dos alguien modifiq algun valor, el segundo ejemplo

x:= x+x

podria ejecutar sentencia uno antes de ejecutar la segnunda sentencia alguien podriaa haceerlo cambiado, x actualizada otro valor.

esa sentencia se divide en tres acc atom de grano fino, cambiar el lado de la primer variable x en un reg

sumarle el segunda x

al resultado almacenar en la pos de meme de X.

nos queda un resultado incorrecto.

ejemp de prog con tres proc, dos modifico el valor de unas variables compartidas y una

2 y 3 si son aciciones de grano fino, si analizamos distitnas historias de este prog concurente, Y y Z van a termianr siempre con los valores y X no, existe un no determinismo ,

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.

• Los valores se cargan en registros, se opera sobre ellos, y luego se

almacenan los resultados en memoria.

• Cada proceso tiene su propio conjunto de registros (context

switching).

• Todo resultado intermedio de evaluar una expresión compleja se

almacena en registros o en memoria privada del proceso.

dos grandes cateroriasde acuerdo a como se comunican los procesos

mem comp tres herram

var comp

semafotos

monitores

hoy variables compartidas, pasaje de mansaje sinc, y asic, rpc (teorico), rendoduv

problemas a resolver: dos problemas fundamentales, seccion criticas, acc de grano grueso, que son hechas por sw, consiste en crear algun protocolo de entrada o salida, y que los procsos puedad ejecutar bloqeus con excl.mutuas

otro problema a resolver es el de barrera, tecnica busy waitntg para resolver estos problemas, se qeuda cehqueando repetidmnete una cond hasta que es verdadera

ventaja: instru de cualq proc

desventaja: nieficiente cuadno hay prog con mas de un rpocesos obre una misma undad de procesamiento, lo v a estar ocupando sin podes avanzar, retrasando la ejec de otro proceso que por ahi es la uqe cambia el valor a verdadero por ej, retrasa el progrrama completo.

si no tenmos q ocmrpartir una unidad de procesamiento por mas de un prog, no es nottoria esta desventaja, como ya definimos,

el problema conssite en poder armar acc atom de grano grueso, por sw, vamos a definir un protocolo de entradac y uno de salida a esa seccion critica, facilmente podemos resolver este problema usando la sentencia await, cuando la utilziamos para excl mutua, prot de entrada < salida >, cualquier soluc a este problema puede usarse para resolver las sentencias await en genral, aquellas incondicionales resolerlas tb, la soluc a este problema, el protocolo de E/S necesita cumplir con

process SC[i=1 to n]

{ while (true)

{ protocolo de entrada; <

sección crítica; SC

protocolo de salida; >

sección no crítica;

}

}

Las soluciones a este problema pueden usarse para implementar sentencias await arbitrarias

**Propiedades:**

**Exclusion mutua** la soluc debe resptar el hechoo de qeu a lo suo un proc puede ejec esa secc critica

**2°ausencia de deadlock**, si dos o más tratan de entrar a una secc crítica y esta libre, uno deberá tener exito, no pueden quedar ahi sin entrr niguno.

**Ausencia de demora necesaria**, si ninguno quiere entrar no debe ser impedido de entrar,

hay uno solo queriendo entrar entonces no debe impedir el protocolo uqe el proc pueda entrar, ejemplo concreto, tienen que entrar de acuerdo al id, primero el uno desp el dos, si o si resp el orden, por uqe hay demora innecesaria? p1, no llego a ese punto, por ahi no empezo, pro el 5 si y esta queirnedo entrar a la sec c , si sigo etsa politica, espera mucho el 5 por ejemplo, distinto es uqe entre los que quieren entrar deje pasar al de menor id.

**eventual entrada**

un proceso que quiere entrar a su sección crítica tiene q entrar en algún momento

tengo uqe asegurar que cuando quiera entrar en la seccion en algun mometno va a poder pasar. Ejemplo, por ej de acuerd al orden alfabético.  **si o si fuertemente fair →** ahi nos asegura que iba a entrar. En principio lo aceptan igual porq es raro el problea, protocolo de entrada o salida de la seccion crítica.

Solucion: que comple con algunas de etas prop, es las sentencias await usadas solo para e.m pero no es una sentencia real es para pseudo codigo, necesitamos resolver esa sentencia, hacer ese protocolo de e/s que repres esos simbolos mayor y menor, si podemos resolver ete problea en la secc crítica.

• Cualquier solución al problema de la SC se puede usar para implementar una

acción atómica incondicional 〈 S;〉 ⇒ SCEnter ; S; SCExit

Reemplazo < por Enter y > por SCExit

Para una acción atómica condicional 〈await (B) S;〉 ⇒ SCEnter ; while (not B) {SCExit; SCEnter;} S; SCExit;

entrar a la sección crítica a través del protocolo de entrada SC enter , leugo chequeo la cond y hasta que no sea verdadera debo seguir iterando pero no dentro de la seccion critica, pero no debemos itrar dentro de la sección cirtica porq sin ons entra ningun proceso q sea el q tiene q modif la cond booleana, entocnes salgo de la seccion critica a traés del prot de salida sc exit y luego volver a entrar par chequear la cond del while. Esto Se va a repetir hasta que B sea verdadera, en ese caso no vamos a salir de la seccon critica directamente continuamos dentro de la seccion critica y ejecutamos las sentencias S, cuadno termina sle con SC exit y se libera para q puedan volver a entrar, entre uqe se encotro y se ejcutan nadie peude generar una interferencia

Si S sentencias fuera vacio entonces await solo como sinc por cndicion para esperar a que una cond se de y ahi cont trabajado, si esa codn fuese cumplle la prop de a lo sumo uan vez se peue hacer por medio de la sentencia while (not B) skip; avanzando hasta qu logro aanzar.

correcta pero ineficiente poqe vemos q en el cuerpo del while estamos contin entrado y saliendo de la seccion cirtica, puede vovler a entrar el mmismo pediddo de entrada que salio y cont iterando de gusto, pdemos mejorarlo poneindo ua demora en que sale y se entra a ala cond critica

Correcto, pero ineficiente: un proceso está spinning continuamente saliendo y entrando a SC hasta que otro altere una variable referenciada en B.

• Para reducir contención de memoria ⇒ SCEnter ; while (not B) {SCExit; Delay; SCEnter;} S; SCExit;

Una primer soluc, es por hw, la idea es deshab las interrupciones, nadie va a poder sacarme, ejec la secc crítica y luego las activo , esto funciona si tuviera una única unidad de proc para todos los procesos. misma unidad de procesam.

process SC1

{ while (true)

{〈await(not in2) in1 = true;〉

sección crítica;

in1 = false;

sección no crítica;

}

}

cumple las 4 pro? ex.m si para ingresar espero en await, ausencia de deadlock, ambos procesos bloqueado en await deberian ambos ser verdaderos, se cumple porque al estar ambos en false en agun momento ingresan,

Ausencia de demora innecesaria: si SC1 está fuera de su SC o terminó, in1 es false; si SC2 está tratando de entrar a SC y no puede, in1 es true; (¬ in1 ∧ in1 = false) ⇒ no hay demora innecesaria.

Eventual Entrada:

• Si SC1 está tratando de entrar a su SC y no puede, SC2 está en SC (in2 es true). Un

proceso que está en SC eventualmente sale → in2 será false, en ese instante y la guarda de SC1 true.

• Análogamente para SC2.

el problema es que una vez que el proc q esta en sc, sale y pone el falso, no significa q el prox es el que está intentando entrar, la unica manera es qeu la politica sea fuertemente fair, asi se garantiza que el proc entra y se asegura la prop. round robin es debil, la eventua lentrada no esta asegurada.

• Si los procesos corren en procesadores iguales y el tiempo de acceso a SC es finito,

las guardas son true con infinita frecuencia.

Se garantiza la eventual entrada con una política de scheduling fuertemente fair.

vamos a ver una soluc con N procesos → cambio de variables

necesitamos una variable q nos diga si la SC esta bloqueada o no. utilizando esa variable reimplementamos la soluc para dos proc basándonos en la variable lock, bloqueo y desblouqueo con esta variable.

process SC [i=1..n]

{ while (true)

{〈await(not lock) lock= true;〉

sección crítica;

lock = false;

sección no crítica;

}

}

se puede generalizar sin problema para n procesos.

problema sigue siendo de grano grueso, primero reolver la sentencia await, sacarla por alguna sent de grano fino.

Objetivo: hacer “atómico” el await de grano grueso.

Idea: usar instrucciones como Test & Set (TS), Fetch & Add (FA) o Compare & Swap, disponibles en la mayoría de los procesadores.

test and set, atomca de grano fino, tener como param un valor booleano, atomicamente guarda en un reg privado el valor act de esa variable luego modif el valor de la variable bool y la pone en true leugo la func completa retorna el q sea habia guardado al entrar. si entra con falso si o si la deja con v y la devuelve en falso, en verdadero la deaja en verdadero y ret verdadero.

el await para n proc,

bool lock=false;

process SC[i=1 to n]

{ while (true)

{ while (TS(lock)) skip ;

sección crítica;

lock = false;

sección no crítica;

}

}

el unico dentro de la secc el proc q ejecuto, cuando sale de vuelta en fals para q otro peuda encontrar el valor en falso y entrar.

**Solución tipo “spin locks”:** los procesos se quedan iterando (spinning) mientras esperan que se limpie lock.

Cumple las 4 propiedades si el scheduling es fuertemente fair. Una política débilmente fair es aceptable (rara vez todos los procesos están simultáneamente tratando de entrar a su SC)

la tres primeras pro seguro se cumplen, la 4 depende de la politica de scheduling.

while (lock) skip;

while (TS(lock))

while (lock) skip;

Memory contention se reduce, pero no desaparece. En particular, cuando lock pasa a false posiblemente todosintenten hacer Test. cuando vemos que es falsa. chequeamos de forma normal ahsta q lock es faslo y cuando encontramso q es falso reicen chequeamos. test and test and set , reicen cuando encontramos falso.

soluciones fair que le dan mas posibilidad a los proceos de poder avanzar, sin necesaitdad de que el scheduling sea fuertemente fear, ordenar, el prim sea realmente el primero.

37.30.

la cuarta prop se cumple, sin necesidad de que sea fuertemente feir.

protocolo de entada a la secc critica y no usa instruccioes epeciales

pero es mas complejo que la otra solucion, una variable de cada proceso, si se esta o está intentando entrar,

24/9

Clase 3- Memoria comaprtida.

el problema es generar los protocolos de entrada y salida al entrar a esa seccion, como administro la entrada y salida a esa seccion, a partir de esto no deberiamos usar las sentencias await.

Spin locks ⇒ no controla el orden de los procesos demorados ⇒ es posible que alguno no entre nunca si el scheduling no es fuertemente fair (race conditions).

ordenar de alguna materia, ver las tres soluciones para ordenar los accesos .

1era

Tie-Breaker

dos procesos, voy a tener dos que para cada uno esta intentnaddo o esta dentro de s.c in 1 in2, cada uno cadno llega su variable en true, chequea que hace el otro proceso, me fij que quiere hacer el otro, si tiene su varaible en false, yo peud entrar, si esta en true problemas, ambos quieren entrar en sc. ¿Cual es el ultimo proc que está intentando entrar a la seccion critica?

nos sirve cuadno intentan entrar al mismo tiempo hay una tercer variable q decide, el qe entra es el que no tiene valor en esa variable, del ultmo que llego, entra el uqe no está en ese algoritmo .

desde le for hasta la llave de abajo para poder acceder una vezz a la seccion critica, for j va compitiendo, complejo siempo confuso

barreras con las variables compartidas, sincron por cond, todos deben esperar hasta que los n proce hayan llegado a tal punto, otra cosa es cuando la barrera se tienen que usar de forma iterativa, cada uno for y al final todos esperan para recien ahi empezar la sig iteración, barrera reutilizable , como hago para poder reutilizar,