Suponiendo el siguiente código para un programa concurrente con dos procesos (**A** y **B**), debemos analizar con que valores puede terminar una variable compartida **x**

int x = 1;

Process A → *1- Load Pos Memoria x, Reg Acumulado*r

{ int y = 5 → *2- Add Pos Memoria y, Reg Acumulador*

x = x + y; → *3- Store Reg Acumulador, Pos Memoria x*

}

Process B → 4- Load Pos Memoria x, Reg Acumulador

{ int z = 3 → 5- Add Pos Memoria z, Reg Acumulador

x = x + z; → 6- Store Reg Acumulador, Pos Memoria x

}

Cuales son las acciones atómicas que forman parte del programa atómica→ nadie podía intervenir, tocar esa variable y producir interferencia ni inconveniente.

En el ejemplo de arriba:

x → variable compartida por dos procesos, inicia con 1.

Dos procesos locales con esa variable x, en el caso de x le sumo el valor y que está inicializado en 5 y en proceso b le sumó z que tiene 5.

Si el proceso A ejecuta toda la suma sin que el proceso b en el medio comience a realizar su suma, no vamos a tener inconveniente y viceversa. El problema es cuando ambos están tratando deshacer la suma de forma simultánea, cada una de esas asignac o sumas, expresiones cada una se va a descomponer en un conjunto de tres acciones atómicas(sentencias de máquina) en este caso la suma, x:=x+y del proceso a se descompone en tres acciones de grano fino, carga el valor que tener en ese momento la variable x en n un registro acumulardor, 2:al valor del reg acum se le suma el valor que está en la pos de mem que rspres a lavar y, paso 3 se guarda el resultado en la pos de memoria x. Cada una de estas acciones atómicas de grano fino se pueden ejecutar sin que se generen interferencias.

proc b similar solo cambia la pos de memoria sobre la cual se trabaja.

int x = 1;

Process A

{ int y = 5

1- Load Pos Memoria x, Reg Acumulador

2- Add Pos Memoria y, Reg Acumulador

3- Store Reg Acumulador, Pos Memoria x

}

Process B

{ int z = 3

4- Load Pos Memoria x, Reg Acumulador

5- Add Pos Memoria z, Reg Acumulador

6- Store Reg Acumulador, Pos Memoria x

}

cuales son las posibles historias y ver cuáles son las que pueden generar dato erróneo. A ver qué ocurre, primero ejecutamos 1,2,3 y despues recien la suma en el proceso b completo, no va a haber interferencia.

En el reg acum de a 6, pero en X tengo 1

el procesos b toma el valor 1, desp toma base el reg con valor 1, el valor de la la acc atom 5 a uno le suma z y almac en su reg el valor 4 , la variable x vale 1, en el reg acum del prco a está el valor 6, en el reg acum del proc b esta 4, el proceso a uarda el valor de su reg acum en paso 3 guarda 6 en x, desp guarda en x.

se produjo un problema, tenemos distintos resultados en la misma variable debería haber dado 9 de ambas formas. nos da cuatro porque el último que se ejecutó fue b. si hubiera sido el último el resultado final era 6.

Esto genera una interferencia un resultado erróneo que no nos sirve. Algunas formas de resolverlo, tratar de resolver el problema en la sección crítica

Los primeros ejercicios son para solucionar problemas de **grano grueso, por exclusión mutua y sincronización por condición.**

Vamos a usar las sentencias *await* , pseudo código para después poder solucionar.

**< await (B); sentencias >**

**Await b condición:** booleana hasta que B no sea verdadera va estar demorado, hasta que B sea verdadera, una vez que es verdadero se ejecuta eso de forma atómica, algo importante la atomicidad comienza al momento que chequea la condición B y la detecta verdadera, desde ahí hasta que se termina de ejecución el conjunto sentencias todo se ejecuta de forma atómica, nadie puede meterse.

Ejecución paso a paso el proceso llega espera acción por exclusión mutua para chequear la condición booleana B, da falso deja la exclusión mutua, luego intenta, chequea nuevamente, si es falsa deja de nuevo, vuelve a acc con exclusión mutua chequea el valor ahora es verdadero entonces se queda en la exclusión mutua y ejecuta el conjunto de sentencias que hay. Y luego si libera la exclusión mutua para que otro pueda acceder.

Nos brinda en esta forma los dos tipos de sincronización, condic(hasta que B true) y exclusión mutua porque el conjunto una vez que se encontró verdadero se va a ejecutar de forma atómica, sin interferencia. Nos permite ejecutar sentencias condicionales e incondicionales.

Únicamente con uno de los dos tipos de sincronización

Forma del await sólo para Exclusión Mutua

< sentencias >

El proceso ejecuta sentencias en forma atómica. Sólo un proceso a la vez puede

estar ejecutando esa “Sección Crítica” (SC).

Se debe tener en cuenta que si hay varios procesos esperando ejecutar la SC y

esta se libera(el que estaba ejecutando término) cualquiera de los que está

esperando accede a usarla con Exclusión Mutua, y NO de acuerdo al orden de

llegada.

Sincronización por exclusión mutua, la única demora es esperar hasta que nadie esté ejecutando, no esperar condición se pone dentro < el conj de sentencias > que se tiene que ejecutar de forma atómica, sección crítica que va a ejecutar el programa. Es como uno puede resolver problema en la sección crítica a tener en cuenta <sección crítica> →a tener en cuenta, no existe orden entre los procesos que esperan a que esa sección o m.u se libere, s tenemos un proc y tenemos varios proc esperando a que se termine el primero terminó y liberó la e.m con la que estaba trabajando cualquier de los que estaba trabajando por esperar va , puede ingresar cualquiera que ingresó , uno solo a la vez pero cualquiera.

Si uno requiere resolver un problema y requiere que esté de acuerdo el orden debería ser a mano, por ejemplo encolarlo.

Sincron por condición, esperar a que una condición booleana sea verdadera, que use una variable compartida, si variable local no pasa nada, demorar el proceso hasta que una condición se cumpla, await condición booleana listo.

Espera acceder con exclusión mutua, condición booleana await y listo, espera acceder con exclusión mutua para chequear la condición, es falsa vuelve a intentar, es verdadera libera y después puede seguir.

Vamos a necesitar al menos una variable compartida,

Ejemplo 2.

Se tiene un salón con cuatro puertas por donde entran los alumnos a un examen. Cada puerta lleva la cuenta de los que entraron por ella y a su vez se lleva la cuenta del total de personas en el salón.

int Total = 0;

Process Puerta[id: 0..3]

{ int Parcial = 0;

while (true)

{ esperar llegada

Parcial = Parcial + 1;

Total= Total+ 1;

}

}

no hay forma de que total sea local, entonces es compartida, todos la incrementan, a su vez c proc va a tener su propia cantidad local. Cuatro procesos puerta, cada uno su propia copia de parcial, y luego suponemos q infinitamente van a estar trabajando estas puertas, increm una ve que la persona pasó.

En un cierto instante puede haber ms de un procesos uqe este increme total, generando que esto nso cree valores incorrectos. El valor total debería ser la suma de cuatro vriable locales parciales, 4 en cada uno son 20 , el problema es que tengo q ejecutar ese increm de la variable total de forma atómica sin error. Que me quedan menos personas en total q las que entraron, necesito usar await para hacer ese incremento de forma atómica. El increm de la variable parcial, se sigue descompone en tres var atom, (reg, +1 y dr de mem tmp es una sentencia atom), se comporta como tal porque es local cada proc tiene su prop variable parcial, ningún otro puede interferir, porq es local, no es necesario que el increm <ponerlo aca> estaria mal es local no necesito sección crítica. Maximizar el uso de variables locales, menos necesito las sentencias await para sección crítica por ende minimizar variables locales.

Todos de forma paralela, en su variable local parcial y cuando todos terminan acumulan los parciales en la var compartida total. <await> en distinto momento. No sirve para ete enunciado sirve si neesito el ototal . cada puerta sabra cuadno no entra mas gente y el valor total nos interesa cuadno todas las peurtas se cerraron.

El incremento de Total no es atómico, ya que (como vimos en el ejemplo 1) se

divide en 3 acciones atómicas de grano grueso. Podría pasar que más de una

puerta intente ejecutar esa sentencia al mismo tiempo, por lo que podría

solaparse algún incremento y esto genera que el resultado de Total sea menor a

que corresponda (Total < suma de los 4 Parciales). Debemos asegurarnos de que

el incremento se haga SI o SI de forma atómica.

Hay un docente que les debe tomar examen oral a 30 alumnos(de a uno a la vez) de acuerdo al orden dado por el Identificador del proceso

Número de proceso, que estructura seguirá este programa? cada uno espera hasta uqe lo llame, desp esperar a que el docente le diga que termino, muy importante: cuando tenemos q interatuar entre dos proce **sincronizar el inicio COMO EL FIN de la operacion,**  como en este caso cuando arranca y cuando termina el examen, sino el docente estaría atendiendo a una persona que ya se retiro, explic de que deberia hacer cada uno de los proc, uno docente 30 alumnos, cada alumn espera hasta q el doc lo llame, una ez q lo llama y luego esperar hasta q el doc le avise q termine, en este caso sabe iqe son 30 veces porque sabe la cantidd de iteracones, para c/u de los alumnos llaama , le toma examen y le avisa que termino de evaluarlo, como iriamos resolviendo cada uno de estos puntos? variable ocmpartida actual, con ID del alumno que el ocente esta evaluando , que hace el proceso alumno para espara a que lo llamen a el, espera hasta que en actual tenga el valor de su id, es su turno entocnes puede pasar a sser evaluado, sentencia await la demoro sinc por condicion

Como hace el proceso docente para llamar a un alumno particular paa ser evaluado, id del alumno atnendio, en la variable compartida actual vamos a poenr le valor dle alumno que nos indica el cuantifica el i, iterador, actual tiene q estar inicializado con un numeor que no sea el id de ninguno de los alumnos por eso -1, ahora el doc, pone actual con ese valor, i lo ponemos dnetor de una seccion critica, usada para exclusión mutua, eso n oes necesario, el unico que o usa es el docente, lo hara n omas para chequear condiciones, para lo unico que lo usan los alumnos es para chequear si llego su turno, en el pero de los caso mientra esta modf el valor actual, en ese momoent a lo sumo tendra que esperar si esta con el siguiente, como es  **modificado solo pro un proceso**  se peude ejectutar de forma normal, no estarian sincronizandose la toma del examen en si, podria haberse ido el alumno,

necesitamos sincronizar cuando el doctermina de toamr el examen y de esa manera se puede retirar, alguna variable uqe nos permita saber , variable compartida listo, el alumno esperar en una sentencia await q listo sea verdadero para avisar al alumno que terminó, esta varaible lsito hay uqe resetearla, sino llama al siguiente y como liso est en true pasa sin terminar de ser evaluado, una vez que el alumno vio el listo en true, atómicamente lo pasa a falso, y el docente a su vez si o i tiene q esperar a que listo sea nuevamente falso para llamar al sig alumno, no es necesario qeu <Listo> este en seccion critia en el proceso del doncente. en la sentnecia await del alumon el listo = falso > tiene que estar dentro del await, Si en una <listo(true)> pero esta parte listo=false no es necesaria.

int Actual =-1

bool Listo = False;

Process Alumno [id: 0..29]

{ <await (Actual == id)>;

//Rinde el examen

<await (Listo)>;

Listo = false;

}

Process Docente

{ for i = 0..29

{ Actual = i

//Toma el examen

Listo = true;

<await (not Listo)>;

}

}

Asi quedaria la solucion al problema!

**Problema 4**

Primero veo si el cajero está ocupado o no, que la cola de espera esté vacía puede que este vacia pero que e cajero esté ocupado. Gran eror!

Si estuviese ocupado el cajero demorarme en la cola de espera, no aniana al fnal, anciana al principio pero solo me encolo si el cajero esta ocupado, si cajero libre persona ocupa cajero, asta q termine de usarlo, si es encolado espera hast q sea su turno, liberar una vez q deja de usarlo, despertar o avisarle al primero de la cola uqe puede pasar, si la cola vacia, liberar cartelito libre ponele, pero desocupado solo si no hay nadie, sino avisa al primero de la cola, el cajero recurso que tienen q usar las personas pero los procesos son las personas, la adm de como los proces persona van a ir accediendo lo hacen entre ellos, ningún coordinador, administra el acceso al recurso entre ellos. **¡chequea recursos compartidos y procesos!**

Cola especial ya que depende de la edad de la persona se encola al final o al principio, la estrucutra existe y la función tb, cola especial, función agregar(elem,edad, id), funcion sacar(id) primer proceso en la cola, de acuerdo a esa estructura implem los proc persna q tienen q usar el rec ocmpartido, vamos a tene la cola vaiaable c, compartida porque todas las personas tiene q interactuar con la misma cola, todos agregan o sacar elemetos en la misma estructura, si dos se posicionan en la misma poscicion se sobreescribe un proc llega antes que el otro pero se encola al revés, necesitamos uqe el trabajo con la cola sea con EM un unico proc agregand oun elem, ademas no puede mietras un elem se esta agregando otro ejec un agregar o un sacar, todo esto <>

Vamos a necesitar await para ejec el agregar con exclusión mutua , como realizo la espera,

una variable compartida que nos indique quien es el proceso persona, demoramos con await hasta que tengamos el valor de la persona correspondiente, tenemos siguiente = -1

primero va a ser nadie, lo uqe hacemos es una sentencia (await siguiente==id)

entonces se demora el proceso aqui cuando sig toma el id es su turno y pasa al cajero, igual que con los alumnos, para liberar el cajero? el valor de sguiete tome el valor del que está primero en la cola, y le pasamos el turno a ese proceso que estaba primero, sacr el primer elem y decir q sig v a tomar ese valor, sig=sacar(c)

al estar trabajando con la misma cola lo tenemos q hacer dentro de una sentencia await EM

que ocurre? que pasa si la cola c esta vacia? si hacemos sacar de la cola vacia, que valor devuelve? a quien le paso el turno? y lo marco como que le cajero esta libre, le vuelvo a poner -1 a sigiente, libre para q el que llegue lo utilice,

colaEspecial C;

int Siguiente = -1;

Process Persona [id: 0..N-1]

{ int edad = ….;

<Agregar(C, edad, id)>;

<await (Siguiente == id)>;

//Usa el cajero

<if (empty(C)) Siguiente = -1

else Siguiente = Sacar(C)>;

}

si esta vacia tal.

PRoblema : que pasa cuando la persona llega y el cajero estaba libre, sig -1? se agrega y se queda esperando a que sig tenga su id , la unica forma de que sig tenga su id es que alguien se retire y lo saque y le de valor, ERROR común , antes de encolar verificar si el cajero está ocupado, si no está ocupado no hay q agregarse a ninguna cola.

me auto despierto si sig = -1 , siguiente soy yo sig=id “me despierto” , sino está ocupado entonces me encolo

colaEspecial C;

int Siguiente = -1;

Process Persona [id: 0..N-1]

{ int edad = ….;

<if (Siguiente = -1) Siguiente = id

else Agregar(C, edad, id)>;

<await (Siguiente == id)>;

//Usa el cajero

<if (empty(C)) **Siguiente = -1**

else Siguiente = Sacar(C)>;

}

**await usada para exclusión mutua, usando el if, de forma atómica, pero de acuerdo al valor de siguiente en ese instante hago una cosa u otra, no me demoro, la demora en el await por sinc por condición**

cuando el orden de llegada hay q cambiarlo es de esta forma orden q no sea de llegada o por id, van a tener q utilizar alguna estructura de este tipo.

Desp alguna soluc de grano fino, utilizar de grano fino sin las sentencias await, utulizando busy waititng hasta q alguno modif un valor, ineficiente poq usa prcesador sin avanzar pero para una soluc de grano fino por ahora no tenemos opcion. analiizar los distintos algo de var comaprtidas analizrlos y ver q peude servirpara los ejeccicons.

Proceso Cordinador::

{ while (true){

<Await (not Empty(C));

siguiente= C.Sacar(); término=false>

// espera hasta que termine de imprimir

<await término; siguiente=-1>

}

}

Proceso Persona::[id: 0..N-1]

{ <Agregar(C,id)>

<await (siguiente == id)>;

Imprimir(documento)

término=true

}