Naiouf

Clase 1y2

Acciones atómicas y sincronización.

Asocia un estado el programa, las variables de ese programa, explícitas, declaradas en el prog pero tb var implícitas, los reg contador del programa, como cambio de un estado al otro, con la ejec de una sentencia. El estado del prog concurrente va a estar dado por la combinacion de estos. Cada sentencia formada por una acc atom.

Accion atom, estados intermedios no visibles, cambio en el estado, intercalado de las acciones atom que ejec, en gral la cantdiad de historias posibles de un progrmaa concu es muy grande, pero no todas son validas, en esta interaccion va a haber historias invalidas.

Historias de manera que no se pierdan valores, ni se repitan, no puedo hacer suppociciones de como va a ejecutarse.

acc atom de grano fino, impl por HW, a=b no es atomica, cargar el valor y luego almacenar el valor. si voy a nivel de maquina serian dos instrucc atomicas de grano fino,   
  
Si yo tengo una expresión e , en un proceso, no ref a una vari ref a otra referenciada por otro proceso,  
• Si una expresión e en un proceso no referencia una variable alterada por otro proceso, la evaluación será atómica, aunque requiera ejecutar varias acciones atómicas de grano fino.   
• Si una asignación x = e en un proceso no referencia ninguna variable alterada por otro proceso, la ejecución de la asignación será atómica.  
  
 que involucra muchas acc atom, pero esas son solo sobre variables propias del proceso, ea acc, la puedo ver como atómica aunque involucre mas de una acción atómica de grano fino porque no estpa afectando al resto de las variables, una sit en la que cada y no de los procesos accede y modifica a cada uno de los procesos,   
  
por mas de que e impliqe modificar a un monotn de variables si es del propio proceso no van a afectar su ejecucion.   
**Prog concu donde c u de los proc tiene sus prop variables y no afecta la ejec de las variables de otros procesos, prog concu con proc disjuntos, esta sit no es lo mas comun en concurrencia, por lo tanto necesito establecer un reuqe mas debil que que los proc no compartan variables.   
Ref critica a una ref a una vari que es modificada por otro proceso, va a ser una ref a una var uqe modif otro de los procesos, critico y una sentencia de asig, de la forma x0 exp**

**safitace la prop de a lo sumo una vez, dos posib:**

**la exp contiene a la sumo una unica ref critica(una var ref por otro proc) y x no es ref por otro proceso,**

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

A lo sumo una variable compartida y esta puede ser ref a lo sumo una vez.

La ejecución va a parecer atómica aunque no lo sea porque la variable compartida va a ser leída/escrita una sola 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

//x ni y son modi por otro proceso!!! cumplen con la prop asv.   
  
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

es modif por otro proceso pero no referenciada.

pero y no es ref por otro proceso.

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  
  
 y es una ref critica, modif en otro proce y además x estaa ref en otro proceso. NO CUMPLE  
  
si tengo ejec de procesos que no comparten variables no hay problemas de inter.

si esto no se da pouedo establecer una cond un poco mas débil que es que los proc cumplan con la prop de a lo sumo una vez, si bien no es atómica puede parecerlo proqeu cumple con la prop.  
Pero podria pasar que no se de ninguna de estas sit, ni disjuntos ni a lo sumo una vez,   
de todas formas puede darse de que necesite ejecutarse todo como una acc atom

para esto surge permite contr una acc atom grueso, como una sec de acc atom de grano fino que aparezca como indivisibles.

<e> me dice que tiene que ser evaluado atomicamente, especificando uqe quiero que la ejecute atómicamente.

exclusion mutua, mientras ejec e nadie me interrumple.   
<await (B) ; s> me permite expresar sincronizacion, sinc por cond, pero ademas es atomica o necesito garantizar que es atomica, s garantica que termina, ademas ningun estado interno de s es visible para los otros proc.

El costo de la implementación es alto.   
  
await s < 0 y dec atomic , Await general: 〈await (s>0) s=s-1;〉 → acc atom condicional.  
o el await solo para e.m 〈x = x + 1; y = y + 1〉 → acciones atomicas incondicionales  
estado en el uqe uno fue incrementado y otro no. no visible para el resto.   
  
Caso particular, Ejemplo await para sincronización por condición:   
*Si B satisface ASV(sino no) porque podria pasar uqe asuma que la cond es falsa cuando por alguna interf se volvio verdadera* , puede implementarse como **busy waiting o spin loop do**  
si la cond satif la prop de als, la puedo implementar como busy waiting.  
  
〈await (count > 0) 〉(not B) → skip od (while (not B);)  
  
propiedad atributo verdadero en cualquier de las historias de la ejecución.   
prop de seguridad: nada malo le va a ocurrir a un proceso. me asegura   
politica de scheduling,   
  
prop de vida, algo bueno va a pasar, no hay deadlock. Dependen de las politicas de schedulin.

una acc atom, en un proceso va a ser elegible si es la prox acc atom que va a ser ejecutada, si tengo varios en un momento determinado podría ver que hay varias acc atom elegible, la politica de scheduling de las elegibles dice cual es la que va a ejecutar, fairnees(justicia) trata de garantizar de que todos los procesos puedan avanzar sin importar que haga el resto,  
Una acción atómica en un proceso es **elegible** si es la próxima acción atómica en el proceso que será ejecutada. Si hay varios procesos ⇒ hay varias acciones atómicas elegibles. Una política de scheduling determina cuál será la próxima en ejecutarse.  
La politics de shceudling depende del so no del lenguaje,   
  
debilm fair, si es incond fair, acc atom incon en algu momento ejecutan, pero adeams todas acc atom cond, q se vuelve

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.  
  
si tengo una acc atom, donde await cond acc, si la cond se vuelve verd y se mantiene hasta q la ve el proces, si ocurre es debilemtne fair, esto solo no alcanza, porque puede pasar q la cond cambie, ademas todas acc atom elegible

como se relaciona fairness con la politica de scheduling, fairness incond, toda acc atom incond, es eleg, eventualm es ejecutada → cualquier acc atom incond q en algun mometo es la prox a ejecutarse en algun momento se va a ejecutar.

acc atom condicionales aparecen fairnes dabil y fuerte.   
Una política de scheduling es débilmente fair si : (1) Es incondicionalmente fair y (2) Toda acción atómica condicional(await cond acc) 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.  
  
si tengo una acc atom, donde es await condo acc, si la cond se vuelve true y no cambia hasta q llega el proceso, ocurre esto es debilm fair, pero no me alcanza para asaegurar que cualquier secuencia await se ejecute.   
 La manejra de resolver esto, es teniendo una fuertemenete fair,   
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.  
  
se vuelve true infitias vveces,entonces en algun momento lo encuentra verdadero.

¿Este programa termina?

bool continue = true, try = false;

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

// 〈await (try) continue = false〉

oc  
  
aseguro que termine con una politica fuertemente fair.

Clase 3 y 4  
  
Problema de la Sección Crítica: implementación de acciones atómicas en software (locks). Barrera: punto de sincronización que todos los procesos deben alcanzar para que cualquier proceso pueda continuar. En la técnica de busy waiting un proceso chequea repetidamente una condición hasta que sea verdadera:   
• Ventaja de implementarse con instrucciones de cualquier procesador.   
• Ineficiente en multiprogramación (cuando varios procesos comparten el procesador y la ejecución es intercalada).   
• Aceptable si cada proceso ejecuta en su procesador.  
  
Cuando los procesos necesitan acceder a un recurso de manera exclusiva.   
  
n procesos tratando de acceder a una seccion critica, para luego ejecutar lo demas no critico. lo importante, lo que llamamos protocolo de etnrada < de salida >, si puedo resolver el problema de la seccion critica,lo puedo usar para resolver cualquier sentencia atomica, ya sea condicional o incondicional.

propiedades -- > condic que son verdaderas cualqueira fuera la historia que tome,

1era exclusion mutua, un proceso solo puede estar en la sc.

2da ausencia de deadlock en la seccion critica, esperando infinitamente por algo uqe no va ocurrir. (livelock todos hacen algo pero no productivo, lo que hacen no les permite avanzar)  
si tengo dos o mas proc intentnado entrar a la sc, es que hay deadlock.  
3era ausencia de dem inn, si no hay nadie y uno quiere entrar entonces el que quiere entrar debo dejar que entre, si la sc esta libre debo darsela.

4ta eventual entrada, tratando de entrar en algun mometno va a entrar, evetualmente entrara.

<sc> como hago para implementar ?

acc atom incondicionales.   
SCEnter ; S; SCExit  
   
  
condicionales, proteger,  
〈await (B) S;〉   
SCEnter ; while (not B) {SCExit; SCEnter;} S; SCExit;  
  
Si S es skip, y B cumple ASV, 〈await (B);〉 puede implementarse por medio de ⇒ while (not B) skip  
intenta nuevamente, si cumple la prop de asv lo hago con ese lazo, espera activa.

ineficiente.

process SC1

{ while (true)

{ in1 = true; # protocolo de entrada

sección crítica;

in1 = false; # protocolo de salida

sección no crítica;

}

}  
  
podria darse que otro entre, s1,s2, verdadero, s2 verdadero, y ahi quedan dos en sc, los procesos no estan chqueando que pasa,   
Solución de “grano grueso”  
esperar a que el otro no esté,  
  
  
process SC1

{ while (true)

{〈await(not in2) in1 = true;〉

sección crítica;

in1 = false;

sección no crítica;

}

}  
si no estprocess SC [i=1..n]

{ while (true)

{〈await(not lock) lock= true;〉

sección crítica;

lock = false;

sección no crítica;

}

}

espero a lock falso y lo pongo en verdadero, solucion de grano grueso para n procesos. com la paso a grano fino?

tratar de hacer etomico este awIAT,   
  
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.   
  
bool TS (bool ok); {   
 < bool inicial = ok;   
 ok = true;   
 return inicial; > }  
variable booleana luego ejecuta atmicamente las acciones, ok que es una variable, esta ejecucion de ts ya me la brinda el procesador de manera atomica, mientras que guarde ok en inicial y retrone, nadie me puede interrumpir,   
tes and set, devolver un valor, lock en verdadero de manera tal que ya pude ser yo que pude entrar cuando ts devuelve falso, puede acceder, ts hasta que da falso, si es true ya había alguien.

Solución tipo “spin locks”: los procesos se quedan iterando (spinning) mientras esperan que se limpie lock.--> queda dando vueltas hasta que alguno logra acceder.

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)

TRATAR de mejorarlo.

TS escribe siempre en lock aunque el valor no cambie ⇒ Mejor **Test-andTest-and-Set**

mejorarlo, cambiar el protocolo y fijarme si el valor de lock es verdadero o no, y si no hay nadie me quedo esperando, veo q no hay nadie trato de ganar el acceso.

Memory contention se reduce, pero no desaparece. En particular, cuando lock pasa a false posiblemente todos intenten hacer TS.

**Tiebraker**

si hay dos procesos que estan llegando al mismo tiempo al llegar, no tengo en cuenta el orden en el cual se fue pidiendo el acceso a la sc, entonces soluc con justicia al acceso a la sc, algoritmo, Tiebraker.

si tengo dos procesos cada proceso cuadno llega ademas de intentar entrar, dice soy el ultimo, el protocolo de entrada dice voy a poder entrar cuadno no haya nadie o el otro es el último, compiten usando una variable adicional, ultimo de manera tal que permite romper el empate, en caso de que dos hayan llegado al mismo tiempo

puede generalizar de dos proc a n porqe cada uno debe pasar por una serie de etapas, e c u debe ganas el acceso a todos, protocolo complejo para su version a n procesos.

otros algoritmos

**Ticket**, la idea es tener un lugar centralizado donde los procesos, como cuando voy a un negocio sacó un número único, y espero hasta que lo llamen, centralizado, cuando crece puede haber problemas!!

centralizado en la variable número, funciona bien para pocos. Desp puedo tener probl de contención de memoria

process SC [i: 1..n]

{ while (true)

{ < turno[i] = numero; numero = numero +1; >

< await turno[i] == proximo; >

sección crítica;

< proximo = proximo + 1; >

sección no crítica;

}  
para n procesos, el proceso i , va a guardarse en turno de i y se guarda el valor de turno, y luego incrementa el valor de número. todo esto de manera atómica.   
Nadie tiene que ver en qué momento aumentó y me llamaron porque pueden robarmelo.

espero a que el num que tengo sea igual al proximo que van a atender, acc atom cond, cuando coincide yo soy el prox, cuando termino libero el acceso para que otro pueda entrar en sc, se va a atender al prox que quiera acceder. El proximo va a ser uno mas.

tres acc atom, 1, y 3 incon, 2 condic, grano grueso, como paso a grano fino??

la primer instrucc:   
< turno[i] = numero; numero = numero +1; > reemplazo con fetch and add.

**fetch and add turno[i] = FA (numero, 1); //mas de una cosa atomicamente, guarda el valor, incrementa el valor y me devuelve el valor inicial. parecido a TS pero var entera.**

await turno de i == proximo, que mi variable coincida con el turno, uqe carác tiene? puedo pasar de esa accion a una de grano fino?   
  
cumple con la prop de a lo sumo una vez, puedo decribirlo con skip, porqe cumple asv,   
while (turno[i] <> proximo)skip;  
  
proximo = proximo + 1;//seria el protocolo de salida , uso del conoc aparece después de que salgo de la seccio crítica, no hay nadie que pueda estar accediendo proque próximo aun no salio, voy a ser el unico uqe puee esatr cambiando el valor de proximo, entonces puedo quitar lso protocolos < >

**Algoritmo Bakery**: Soluc un poco mas distribuida.   
No usa esa var global número, llegan y se fijan cual es el mayor número y se autoasigna un número mayor que de cualquiera de los que esta esperando,no hay un lugar centralizado, miro que psa con el resto, nusco y me autoasgino, cada proceso que trata de ingresar recorre los números de los demás y se auto asigna uno mayor. Luego espera a que su número sea el menor de los que esperan. Los procesos se chequean entre ellos y no contra un global.

mayor dificultad, Implica buscar el máximo, no es sencillo,

El algoritmo Bakery es más complejo, pero es fair y no requiere instrucciones especiales.

• No requiere un contador global proximo que se “entrega” a cada proceso al llegar a la SC.

**Otro problema a resolver con variables compartidas**

**Sincronización barrier:** una barrera es un punto de demora a la que deben llegar todos los procesos antes de permitirles pasar y continuar su ejecución.   
Hasta que no hayan llegado todos hasta ese punto no puedo avanzar.

Dependiendo de la aplicación las barreras pueden necesitar reutilizarse más de una vez (por ejemplo en algoritmos iterativos).

podría usar un co, pero si lo que tengo que usar es iterativo, se destruirían cada vuelta los procesos, toma tiempo de procesador.   
crear los procesos y desp tratar de sincronizarlos.

Si tengo que lograr que n procesos lleguen a un punto para luego seguir, uso una variable compartida, opción centralizada funciona bien para pocos. C proc cuando llega sumo uno y se que los n llegaron-

entonces, como puedo tratar de res esto? usando n vairables,

int arribo[1:n] = ([n] 0), continuar[1:n] = ([n] 0);

process Worker[i=1 to n]

{ while (true)

{ código para implementar la tarea i;

arribo[i] = 1;

〈 await (continuar[i] == 1); 〉

continuar[i] = 0;

}

}

process Coordinador

{ while (true)

{ for [i = 1 to n]

{ 〈 await (arribo[i] == 1); 〉

arribo[i] = 0;

}

for [i = 1 to n] continuar[i] = 1;

}

}

ineficiente.

int arribo[1:n] = ([n] 0), continuar[1:n] = ([n] 0);

process Worker[i=1 to n]

{ while (true)

{ código para implementar la tarea i;

arribo[i] = 1;

while (continuar[i] == 0) skip;

continuar[i] = 0;

}

}

process Coordinador

{ while (true)

{ for [i = 1 to n]

{ while (arribo[i] == 0) skip;

arribo[i] = 0;

}

for [i = 1 to n] continuar[i] = 1;

}

}

lo puedo hacer con un proceso coordinador, esperando a que c u de los procesos avise q ha llegado, los workers c vez q llegan poner arribo y luego quedan esperando por una variable va a cambiar cuando haya visto que todos llegaron, continuar de i , estas sentencias arribo y coordinar, cumplen con asv entonces puedo convertir en sentencias, await,

esperan sobre una única variable.  
  
Solucion para que cada worker tb sea coordinador, organziarlos en forma de arbol, si plateo esto, cuando un proceso llega a la barrera le avisa a su padre que cuadno ve que sus dos proc hijso llegan le avisan a sus padres, van bajando hasta que lelgaron a la barrera, cuando la raiz recibe el aviso, entonces todos llegaron entonces pueden seguir, pueden cont desde arriba hacia abajo, cuadno n crece es mas eficienete.

coordinadores para los hijos, pero workers para los padres,   
no hacen todos los procesos lo mismo.

• Combinar las acciones de Workers y Coordinador, haciendo que cada Worker sea también Coordinador.   
• Por ejemplo, Workers en forma de árbol: las señales de arribo van hacia arriba en el árbol, y las de continuar hacia abajo ⇒ **combining tree** barrier(más eficiente para n grande).  
  
• Una Barrera Simétrica para n procesos se construye a partir de pares de barreras simples para dos procesos:

la barrera mas simple es una para dos procesos, se esperan mutuamente.

i espera a j

y j espera a i

esa seria la barrera mas simple q puedo tener, como puedo utilizar barreras para dos procesos, y combinarla para n procesos.

**butterfly barrier,** si tengo n procesos, n potencia de dos, puedo pensar uan soluc en la cual los proc pasen por distintas etapas, en cada etapa un proceso worker sincroniza con uno diferente, cada workear sincroniza con un proceso.

a distancia 1 por ejemplo, desp sincronizar con otro que está a distancia dos.

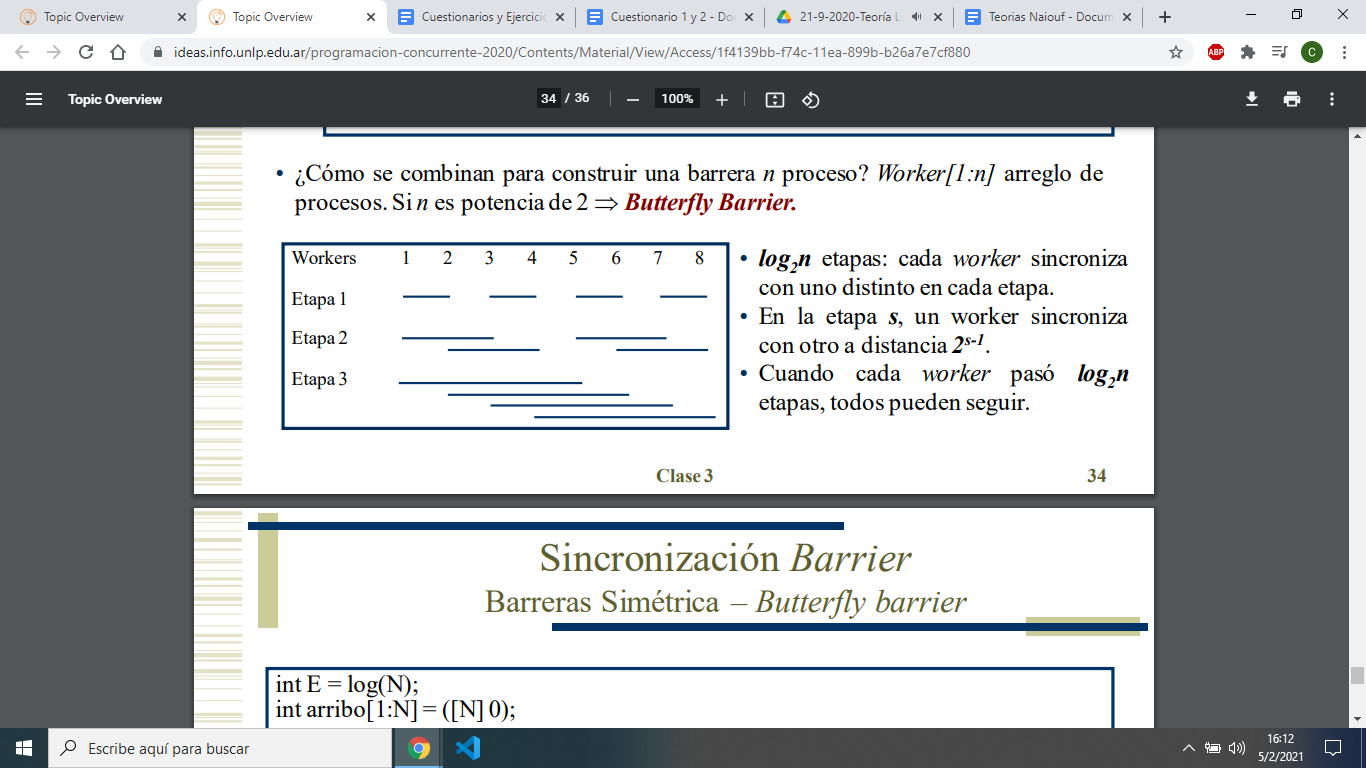
• ¿Cómo se combinan para construir una barrera n proceso? Worker[1:n] arreglo de procesos. Si n es potencia de 2 ⇒ Butterfly Barrier.

Workers 1 2 3 4 5 6 7 8 Etapa 1 Etapa 2 Etapa 3

• log2n etapas: cada worker sincroniza con uno distinto en cada etapa.

• En la etapa s, un worker sincroniza con otro a distancia 2s-1.

• Cuando cada worker pasó log2n etapas, todos pueden seguir



todos pudieron sincronizar con todos, en la etapa 3 el uno con el 5 que sicnronizo con los otros y asi.

si duplico la cantidad de procesos de 8 a 16, cuatro etapas.

32 - 5 etapas.   
  
En butterfly me alcanza con codificar un procesos, porque todos hacen lo mismo.   
gano que la complejidad en tiempo va a ser logarítmica, mejor q lineal.

si tengo pocos procesos, centralizada si me alcanza para sincronizarlos.

si tengo mas procesos solución simétrica.

Si es impar en butterfly, procesos ficticios que sirvan para sincronizar.

Tener distintos tipos de procesos afecta el fairness, no necesariamente, podria hacerlo, afecta a la complejidad de la programacion, tres dijeferenes   
  
los procesos son todos iguales.

arbol los procesos pueden hacer diferentes cosas.

Protocolos “busy-waiting”: complejos y sin clara separación entre variables de sincronización y las usadas para computar resultados. no hay una separación clara.

Es difícil diseñar para probar corrección. Incluso la verificación es compleja cuando se incrementa el número de procesos. //sobretodo cuando la cant de procesos crece.

Es una técnica ineficiente si se la utiliza en multiprogramación. Un procesador ejecutando un proceso spinning puede ser usado de manera más productiva por otro proceso. tiende a ser ineficiente.   
  
busy waitng,adecuadas cuadno acada proc tiene su propio procesador.   
  
necesario que uno trate de encotnrar herram qe sean un poco mas adeacuadas para lograr la sinc.

Clase 3 SEMÁFOROS.

Uno de los que esté esperando, no necesariamente el primero.  
sem free= 1;   
process SC[i=1 to n] {  
 while (true) {   
 P(free);   
 sección crítica;   
 V(free);   
 sección no crítica; } }  
si free es cero ninguno podría entrar a sc.   
El incremento de la variable semaforo ya va a ser atomico.

Ventaja, solo puedo accederlo a través de operaciones, con la ventaja de que estas dos operaciones ya son atomicas, comparado a las variables compartidas.

Semáforo ⇒ instancia de un tipo de datos abstracto (o un objeto) con sólo 2

operaciones (métodos) atómicas:P y V.

Internamente el valor de un semáforo es un entero no negativo:

**• V → Señala la ocurrencia de un evento (incrementa el valore del semafoto) .**

**• P → Se usa para demorar un proceso hasta que ocurra un evento (decrementa el valor y si no es positivo se demora, resolver problemas sencillamente comparado a variables compartidas).**

Generales:

la operacion v sobre el semaforo es un incr atomico,  
• Semáforo general (o counting semaphore)

P(s): 〈 await (s > 0) s = s-1; 〉 necesito que el valor del semaforo sea positivo, el proceso debe demorarse hasta qeu el valor de la vairable sea positivo.  
si un proceso llega a p y no es positivo, se duerme en alguna estruc en donde duermen los que esperan, entonces una vez que se duerme se queda ahí hasta que el valor de ese sem cambie y sea despertado. Despierto cuando ocurre V. **No me paso chequeando por la condicion.** No necesariamente el que primero duerma es el que primero se despierte, peude o no ser una cola

V(s): 〈 s = s+1; 〉   
  
tanto el incremento como el incremento pueden causar demora.

**En el semáforo binario, tanto el inc como el dec pueden causar demoras**

Va a ser general o binario depende de como lo use.

Barrera. Procesos que se esperan de manera tal, que hasta que todos no llegue a ese punto no pueda continuar.

Problemas de barreras para dos proceso, lo que importa es cuando llego y cuando sale. C vez que llegue avisa que llegó.

como resuelvo el problema de barreras con semáforos, los problemas de barreras consisten en procesos que se esperan en algún punto que si no llegan a este punto no pueden seguir, combinar para tener barreras para n rpcoesos.   
Lo que importa es cuando un proceso llega a una barrera y cuadn osale. y luego espere a que llegue el otro.  
  
que pasa si los procesos hacen v y luego v? se produce deadlock!!  
  
Productores y consumidores.

SPLIT: 0 ≤ b1, + ... + bn ≤ 1 binarios forman un sem bin div, si la suma de todos los semaforos esta entre 0 y 1.   
Semaforos binarios divididos, 0 o 1, si la suma de los semaforos está entre 0 y 1.   
Los puedo ver como un unico semaforo binario.   
Van a servir para implem **excl mutua**, me asegura que lo q esta entre un sem y otro se ejecuta con em. Va a empezar con una op   
• Importantes por la forma en que pueden usarse para implementar EM (en general la ejecución de los procesos inicia con un P sobre un semáforo y termina con un V sobre otro de ellos).

• Las sentencias entre el P y el V ejecutan con exclusión mutua

En que consiste el tema de prod y consu?

Espacio compartido de memoria, EL buffer es una secc critica, entonces necesito acceder a él con em, tengo que ver que este vacio, p sobre sem vacío inic en 1, estado inic vacío, permitirle entrar a un productor,

p vacio

accedo a sc

pogno los datos

y aviso q esta lleno.

consumidores, veo si lleno y cuando sale aviso que está vacío, si llegan dos productores de forma consecutiva, el semáforo vacío va a estar en 0 entonces va a estar dormido hasta que alguien se retire.

typeT buf; sem vacio = 1, lleno = 0;

process Productor [i = 1 to M]

{ while(true)

{ …

producir mensaje datos

P(vacio); buf = datos; V(lleno); #depositar

}

}

process Consumidor[j = 1 to N]

{ while(true)

{ P(lleno); resultado = buf; V(vacio); #retirar

consumir mensaje resultado

…

}

}  
  
Vacío y lleno son un semáforo binario dividido, lo que ocurre entre ellos dos va a ser con em.

por que dos semáforos y no uno?

por que no alcanza con un solo semáforo?? porque los procesos esperan por diferentes condiciones, si bien no explicita a que conficion pertenece implicitamente sabemos para que sirve.

Hay dos grupos, competencias entre procesos de un mismo tipo y de distinto tipo, no alcanza con uno solo y un v sobre el mismo, porque los procesos esperan por dif condiciones, semáforos distintos corresponden acciones distintas.

Que pasa si tengo un buffer mas grande? lo veo como una cola.. ambos sacan o depositan del primer lugar. Único proceso productor y único consumidor.

Necesito proteger el acceso a las variables compartidas, com v mutex, y p mutex, cada pos del buffer se convierte en sc, un p sobre el sem y un v, dos niveles de exclusión, me va a dejar entrar si hay lugar y luego va a chequear que yo sea al unico que teng q dejar un dato   
  
Cuando compiten por conjunto de recursos, parcialmente superpuestos, adq un recy lo bloquee y asi sucesivamente, cada uno bloquea y entonces se produce deadlock   
  
una cond necesaria para que haya deadlock, es la espera circular, si nunca se llega a la espera circular haciendo que un proceso haga las cosas al revés que los demás, si un filosofo

Si un filo, tiene dos vecinos puede ser que uno nunca coma, si bien no hay deadlock podria llegar a provocarse inanición. Nunca termina de hacer lo que necesita. Pero se asume que los procesos van a estar mas tiempo pensando que comiendo.

**La solución con semáforos ya es de grano fino.   
exclusion mutua selectiva, compiten un conjunto de procesos con otros.**

Lectores y escritores, se adueñan de la bd, los lectores necesitan bloquear a los escritores pero solo el primer lector necesita tomar el acceso, el ultimo lector debería liberar el acceso,

cuando llegan los lectores, suma uno y solo si   
Los lectores no compiten con otros lectores pero si con los escritores.

< nr = nr + 1; if (nr == 1) P(rw); >

lee la BD;

< nr = nr - 1; if (nr == 0) V(rw); >  
resuelve el problema, pero tiene un problema es que le doy pref a los lectores-  
  
sincronizacion por condición, passing the baton, implem cualquier await y ademas controlar el orden o el tipo de procesos sean despetados, contar cuantos procesos de cada clase estan tratando de acceder a la bd, para que un escritor acceda, nr y nw sean 0, y si soy lector aunque haya otro lector no importa, los lectores chequean que no haya escritor, accion atomica condicional,   
  
una condición se superpone parcialmente con la otra, implícitamente yo se que condicion espera cada, esta sit no hay un semáforo que pueda discriminarla, entonces uso **passing the baton** para tratar de resolver los await generales, la idea es pensar a los problemas tipo de las carreras de posta, el qe tiene el baton tiene el permiso para hacer algo, permiso para eject, y cuando dejar el acceso a la sec crit, le pasa el control a otro proceso, lo que esta esperando para acceder, si ninguno esta esperando deja el baton libre para el próximo, excl mutua y por cond y ademas me da control control sobre a quien le doy el permiso.

passing the baton, cuando llego a una soluc de grano grueso, mi soluc está escrita en término de sentencias atom condicionales y sentencias atom incondicionales.   
  
La sincronización se expresa con sentencias atómicas de la forma:   
F1 : 〈 Si 〉 → incondicionaels  
F2 : 〈 await (Bj ) Sj 〉 Puede hacerse con semáforos binarios divididos (SBS). → condicionales.   
  
Al estar entre corchetes lo que está dentro lo ejecute con e.m, un semáforo binario que me permita implementar, voy a usar un sem

e semáforo binario inicialmente 1(controla la entrada a sentencias atómicas).

Utilizamos un semáforo bj y un contador dj cada uno con guarda diferente Bj ; todosinicialmente 0.

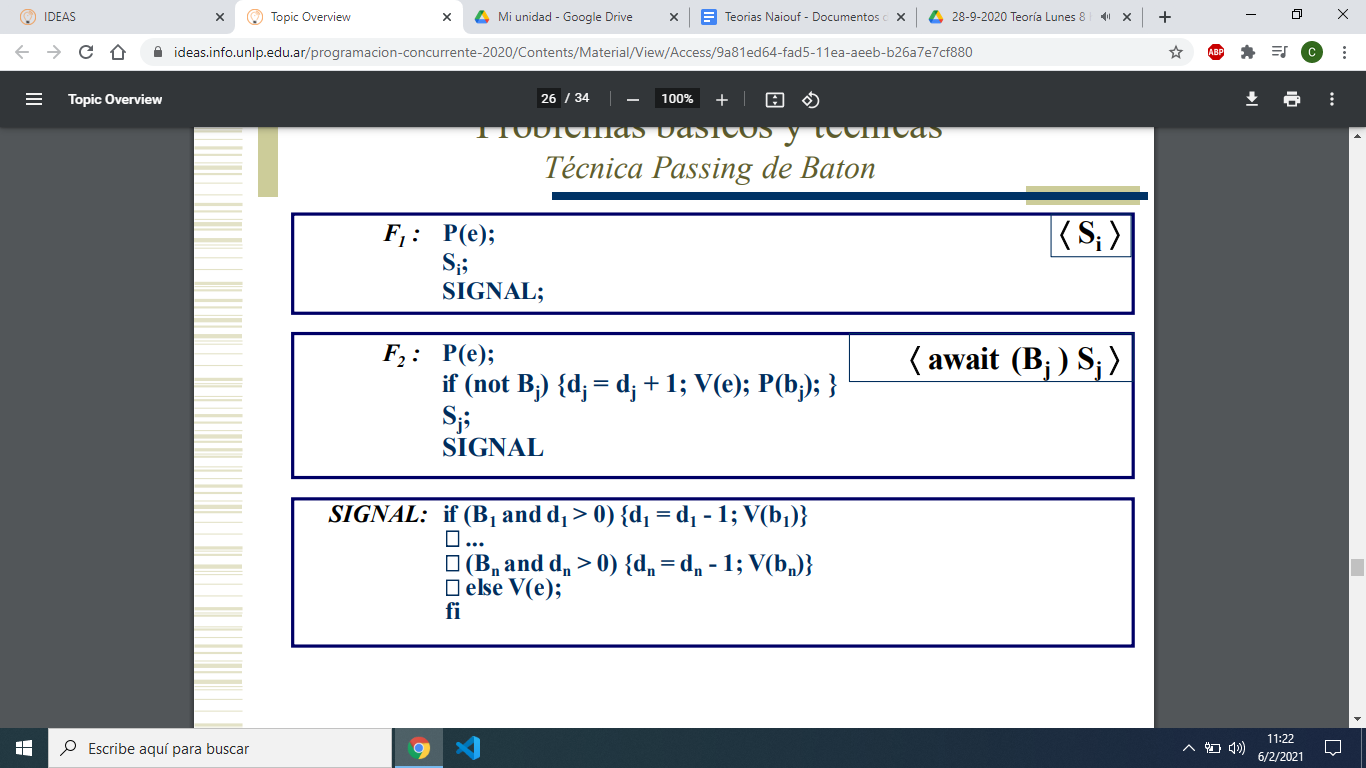
tengo uqe indetificar las condic dif por las uqe esperan los semaforos,

bj se usa para demorar procesos esperando que Bj sea true.

dj es un contador del número de procesos demoradossobre bj.

e y los bj se usan para formar un SBS: a lo sumo uno a la vez es 1, y cada

camino de ejecución empieza con un P y termina con un único V  
  
para formar un semafo binario dividido, todo lo q se ejec en el medio e.m



F1:trato de acceder de forma exclusiva, una vez q lo logro ejecuto y luego libero.

F2: trato de ganar el acc excl, cuando entro me fijo como es la condicion si bj es true, puedo entrar a ejec el conj de sentencias Pj, entonces me voy a dormir en el semaforo q controla la cond, libero el acc exclusivo y me duermo, cuanto que va a haber uno mas, libero el acc excl y luego me duermo. si.  
signal le da prioridad a los que ya estaban en el sistema

de todas las condic me queda con una y de esas la q sea verdadera, los q ya estaban tienen mas prioridad, permite resolver sit en la cuales no importa cuales son las condic, **alcanza con ver cuales son, cuantas tengo, es una técnica general para resovler sinc por condición**

No me alcanza con despertar a un proceso, tengo que saber a cual a veces, passing the baton combinarlo con un semáforo privado,   
   
Monitores, brindan un mecanismo de abstracción, más estructuras dentro lo que es la prog de manera tal que puede encapsular la repres de los recursos y brindar un conj de operaciones que sean las que me permitan manipu el recurso, tipo abstracto de datos, me abstraigo y lo que interesa es que interesa para manip, pero puede ser utilizado por processo de forma concurrente, proced que implementan las operaciones sobre estos recursos, la exclusion mutua en el caso de los monitores la em aparece de manera implícita, no pueden estar ejec al mismo tiempo mas de un proceso, dentro de un monitor, no me preocupo porque haya em no va a haber dos ejecutando dentro del monitor,la sincro por condicion se dan por variables condición, demorar los procesos y despertarlos, lo invoco con el nombre, realizar un llamada a una oper dentro de ese monitor, lo único visible para el proceso son las repres que el monitor me brinda, no conoce el estado ni la representa, solo como lo puedo acceder, y esto me lo da el monitor.

variables permanentes que me representan lo que estoy representando, luego procedimientos que pueden ser invocados desde el monitor, Las variables permanente dentro del monitor. No necesito proteger el acceso porque soy el único.   
Adeams la sinc por condicion la tengo que programar explicitamente,   
  
nombre, monitor, parámetros, el proceso que realiza la invocación, cuando hace el call va a quedar demorado hasta que sea tomada, ejecutada y se devuelvan los resultados. Retoma el control en la instruc siguiente.   
8. 5 proc empleado,   
como no hay dos ejecutando a la vez, se va a quedar esperando, por eso no necesito preocuparme por brindar la em sino que es implicita, las variables dentro del monitor, no necesito proteger el acceso a cant,   
  
**MONITORES SINCRONIZACION POR CONDICION**

variable condición  
cuando quiero hacer dormir a un proceso porque la cond por la que espera no es verdadera, espero por un recurso , lo mando a dormir con una var condicion, libera el acceso, signal me permite despertar al proceso y sacarlo de la cola, a través de una operacion **WAIT***signal permite desperta el primero y lo saca de la cola, el proceso va a poder ejecutar cuadno adquiere el acceso*

*signal all → despierta a todos.*

empty, decirme si hay una cola controlada por la variable condición que está vacía o no,   
• wait(cv, rank) → el proceso se demora en la cola de cv en orden ascendente de acuerdo al parámetro rank y deja el acceso exclusivo al monitor  
unrank cuando dormimos el proceso q dormimos va a parar al final de esa cola y lo puedo agarrar ordenado de acuerdo al valor de ese proceso, no utilizamos estas op en la P.   
• minrank(cv) → función que retorna el mínimo ranking de demora

**• Signal and Continued:** el proceso que hace el signal continua usando el monitor, y el proceso despertado pasa a competir por acceder nuevamente al monitor para continuar con su ejecución (en la instrucción que lógicamente le sigue al wait).  
a la instrucción siguiente.  
**• Signal and Wait:** el proceso que hace el signal pasa a competir por acceder nuevamente al monitor, mientras que el proceso despertado pasa a ejecutar dentro del monitor a partir de instrucción que lógicamente le sigue al wait.  
  
que pasa cuando uno despierta a otro a partir de la oper signal, signal and wait, un proceso ejecutando en el monitor, se va a ir a esperar por el acceso y el que tiene el signal va a ir a ejecutar, cualquiera puede acceder, no necesariamente le primero.   
24-25

**wait siempre se va a dormir, p solo si vale 0**

**signal si hay proc dormidos depierta al primero, si no hay nadie no despierta a nadie, V siempre se incremen 1.**

el que se fija si hay dato o no es el monitor.   
  
passsing the conditions, si quiero tener un orden, simular un sem donde los procesos se ejecuten en el orden en que fueron enviados, lo que cambia es la impl de esto, el proceso P,

si s es 0 wait( A DORMIR)

sino decremento.

En el momento en uqe un proced va a cambiar una cond la cambiar si hay alguien esperando, solo increm si no hay nadie, pasar la condición a aquellos procesos que ya estaban esperando, tienen prioridad los procesos que ya estaban dormidos esperando, compiten no deterministicamente.

no hacer visitable uqe la condición cambio para aquellos uqe están llegando, no ven que se modifico poruqe

procedure V ()

{ if (espera == 0 ) s = s+1

else { espera --; signal(pos);}

};

**passing the condition es fundamental en el caso de los monitores, me permite tener un orden en que los procesos se van a ir ejecutando  
una vez que llegue a la solución de grano grueso, aplicando esta técnica podría usarla siempre.**reemplazar empty por una cola de espera, cambio empty por su implementación.

semáforo privado → un único proceso se puede dormir

s es un semáforo privado si exactamente un proceso ejecuta operaciones P sobre s.

Resultan útiles para señalar procesos individuales. Los semáforos b[id] son de este

tipo.

26-admin o alocación shortest first.

Tratar de admin un recursos, utilizando el proximo es el uqe va a utilizar el rec por menos tiempo

pido un recurso con un request, y lo devuelvo con un realice.

Cuanto tiempo quiero usan ese recurso?

Este, puede estar libre o ocupado, la representación con una varaible logica. y los procesos pueden tener que dormirse sobre una variable condicion cuando no esten libres,

si hay un proceso que invoca

si esta libre en falso, sino va a mandar a dormir a ese proceso, con el valor de tiempo que es el parametro, wait con prioridad.

si no hay nadie, libera lo pine en true, y si hay lo despierta signal. Pero no hace visible la cond de que el recurso está libre  
  
cuanod uso passsin the condiction no necesito poner a los procesos en un lazo,

que hago si no hago wait con prioridad,

27, sin utilizar wait con prio, request si esta libre libre en falso, sino lo inserta y luego lo duerme, en vez de haber una variable cond, hay un arreglo, lo duermo sobre una variable cond turno de ese proceso.

ese sacar podría ser dfi dependiendo de cuál es la op que quiero llevar a cabo.

28. hay una solución al problema del buffer limit, lo q tengo son las imple de depositar y reirar.

aca no hay que controlarlo como no tengo dos procesos en el monitor, no hay dos haciendo depositar.

Lectores y escritores  
Varios procesos lectores,, cual es el recurso que abstrae el monitor?

si tratan de acceder a una bd, uno piensa que el monitor abstrae la bd, lo que pasa es que no podria tener mas de un lector leyenedo en la bd, el monitor no abstrae el recurso en si sino que el ACCESO al recurso, en este caso la bd.

Entonces lo uqe ocurre es uqe los lectores va a tener qe hacer un pedido, al leeer y cuando liberen , las operaciones pedido leer, libera leer, cuando haya escritores, al proceso leer a dormir, cuando un lector deja la bd, podria habilitar, para uno o muchos, que ocurre? ok escribir

signal all leer, no se cual, va a ingresar si es un escritor, escribe si desp viene lector,

CLASE 7  
Programación concurrente en memoria distribuida   
Procesadores a donde cada uno tiene su mem local, y hay una red que conecta a estos procesadores.   
programa desitibuido concurrente pero comunicado por mensajes, interface con el sistema de comunicaciones, a treavés de la primitiva a de pasaje de mensajes, canales fisicos o logicos, dif de trabajar con la mem compartida, no debo protegerlo porque automaticamente me va a asegurar que no va a haber dos accediendo al mismo, a partir de estas vairantes se generen los mecanismos, de pasaje de mensajes, relacion entre los que envían y reciben, mailbox, canal con varios posibles emisores, y varios posibles receptores  
unpit, puerto de entrada, varios emisores pero u unico receptor, .

link o enclace punto a punto, un receptor y emisor.

unidireccionales, un sentido y ambos sentidos.

sicronicos o asincronicos, los canales van a ser lo unico que comapartan los procesos.

el acceso a los canales no genera intert, mensajes a traves de los canales, y comunicados a través de recibir/enviar mensajes.

distintos **mecanismos de pasajes de mensajes,** asincr , sincr, remotos o dnu, con dif caracteristicas uqe lo van a hacer mas adecuadso para determinados problemas.

la comunic entre procesos, **patron de intecccion**,

clientes y servidores, brindar serv y un conj los usa.

pares que interactuan, conj de proc que hacen lo mismo y cada cierto periodo se comunican.   
  
mecanismos equivalentes, si con uno se puede hacer con otro también, difiere en que va a ser mas conveniente usar tal.   
  
Pasaje de mensajes ASONCRONICOs, cola de mensajes uqe fueron enviados en un canal peor aun no fueron enviados, id y tipos, canal entrada a través del cual envio y recibo de a un caracter, en ese canal cada mensaje compuesto por un carcter,

Declaraciónde canales → chan ch (id1 : tipo1, ... , idn : tipon )

• chan entrada (char);

• chan acceso\_disco (INT cilindro, INT bloque, INT cant, CHAR\* buffer);

• chan resultado[n] (INT);

Operación Send → un proceso agrega un mensaje al final de la cola (“ilimitada”) de un canal ejecutando un send, que no bloquea al emisor: send ch(expr1, ... , exprn);

al final de la cola de mensajes, esa op no bloquea al proceso que lo envio, no necesito q sea tomado por el receptor, por eso asinc.

Operación Receive → un proceso recibe un mensaje desde un canal con receive, que demora (“bloquea”) al receptor hasta que en el canal haya al menos un mensaje; luego toma el primero y lo almacena en variables locales: receive ch(var1, ... , varn);

Si hay algun mensaje, saca el primero de los mensajes y puede seguir ejecutando, si no hay ningún mensaje se demora hasta que haya al menos un mensaje, puede ser bloqueante porque puede haber demora, queda ahí esperando hasta que apareca algo en el canal.

en el caso de los mensajes asinc, atómicos(cuando hay un proceso haciendo sendo recive no hay otro haciendo lo mimso) y es una cola, los canales son ilimitados, pueden dejar mensajes sin esperar, los mensajes no se pierden ni se modifican, el mecanismo es como si tuviera un semaforo antes y otro desp.

si hay dos que estan tratando de dejar un mensaje en el canal primero uno desp el otro. Ademas de estas dos operaciones ,de enviar y recibir ,está empty si la cola esta vacia o no. si no hay nada en el canal, hacerla, puedo hacer otra cosa mientras.

cuidado en el uso de empty porque puede darse que no termine teniendo el efecto que esperaba, chequeo si el canal está vacío o no, pero entre que cheque e hice la operación de recibir, otro tomó el mensaje, entonces me demoro.

chequeo me dice que esta vacia y me dice que si alguien puso un mensaje entonces pierdo el mensaje.

carac a linea es un ejemplo de los procesos **filtros**, recibe una entrada, transformarla de alguna manera y la saca por un canal de salida.

Otro ejemplo, ordenar 12 lista de números ascendentemente, podria pensar en tener un prcoeso filtro donde recibe num desordenadors y salida ordenados, sort recibe , los ordena y envía ordenados por un canal de salida, de concurrencia tiene poco sacando que estoy dejando en canal y tomando de ahi, pero es secuencial, como sabe sort sabe qe recibio todos los numeros, o podría terminar con un valor especial corte zzz porej.

Pensamos una solucion mas eficiente, construida como una red de procesos que ejecuten en paralelo, armar la salida de forma ordenada, tomar dos entradas e ir mezclandolas, dos listar en una de todos los elementos, con mensajes asincronicos uno puede pensar canales de entrada por cada uno de los de salisda y un carac de especial q va a cerrar cada lista parcial ordenada. La idea es uqe este proceso merge desde dos canales de entrada y va a sacar por out, recibe scuencias ordenadas y saca la secuencia que entro ordenadas.

in1, in2, valor 1 y por inout el menor.

El proceso merge haria algo

chan in1(int), in2(int), out(int);

Process Merge

{ int v1, v2;

receive in1(v1);

receive in2(v2);

while (v1 ≠ EOS) and (v2 ≠ EOS)

{ if (v1 ≤ v2) { send out(v1); receive in1(v1); }

else { send out(v2); receive in2(v2); }

}

if (v1 == EOS) while (v2 ≠ EOS) {send out(v2); receive in2(v2);}

else while (v1 ≠ EOS) {send out(v1); receive in1(v1);}

send out(EOS);

}

va a buscar a un canal y al otro, saca por aou por el menor y saca y va a buscar otro al canal del que salió sigue sacando.

un proceso que recibe desde dos y saca por uno, las dos secuencias ya están ordenadas, a partir de este como hago para tener varias instancias y combinarlas y además como se uqe las secuencias están ordenadas?

cual es la ordenada más corta? tamaño 1, procesos que trabajan en paralelo en dif etapas, procesos filtro, patrón de interacción. Productores y consumidores.

**Clientes y servidores**, servidor maneja pedidos de otros o request de otros, com puedo hacer para implementar una interacción, con mensajes asincrónicos, dualidad entre monitores y pasaje de mensajes, por que? Básicamente un monitor es un manejador de recursos, estado y procedimientos para acceder a un recurso, procesos activos en vez de conjuntos pasivos de procedimientos, los procedimientos se despiertan cuando son invocados, como puedo hacer para simular monitores usando mensajes asincrónicos.

Como hace un proceso cliente para realizar un pedido, poner un mensaje en un canal de requerimientos, donde se van encolando, los pedidos a una canal general y los otros y espera el resultado, canal de rta que solo acceda el, necesito que las respuestas no vayan a un canal general para que sea propia, la forma de imple cliente servidor, van a parar a un canal común donde van todos los pedidos, pero

Los pedidos que hacen los clientes a un canal común, las rtas desde el servidor a los clietnes propios o privados de c u de los procesos.

Varios servidores y que sea tomado por el primero que esté libre.

Un servidor que simula un monitor uqe tiene una unica operacion que brinda el servidor,

servidor while true, ejec infinita, clietne y valroes de entrada y luego va a contestar por el canal rta, los resultados, esta op de send es la se va a aparear con recive.

Que pasa si en lugar de tener un servidor uqe brinda una unica op, brindara varias operaciones?

operación y argumentos! En el caso del canal de request esta agregando una clase de operacion, ademas de decir quien es va a tener que decir que op es la que quiere, depende lo que le pidan ejecutar.

Si lo que hago es simular un monitor, acá faltaría algo!! Faltaria → es que cuando recibo un pedido y no peudo terminar de completarlo, neceisto dormir los procesos para qu cuadno tegna la respuesta darselas, si recibo un pedido, lo demoro hasta uqe tenga lo necesario para devolverle los resultados, no tengo aca wait o signal , pero puedo simularlo, guardo el pedido, lo mando a dormir para poder retomarlo luego, pero el wait lo uqe hace es guardar un proceso con una cola, y cuando cumlpa una condicioón lo saca.

Pedidos de adquirir(si no hay unid dispo, a dormir, sino -1) y de liberar(volver a poner esa unidad en el conj de unidades, si no hay nadie esperando rec disp sino desp a uno de los que estaba esperando. ← con monitores

Con mensajes asincrínicos?? *guardar el pedido y diferir la rta, cuando una unidad sea liberada, agarrar un pedido y mandar la unidad que corresponda, el proceso cliente, envía un pedido por el canal request, estoy i, quiero una unidad, un vez que hizo el pedido cual es el canal que puede usar, si la operación es adquirir o es liberar, si tegno unidades disp disp > 0 , cuál es la que puedo usar, saco la canti y contesto al cliente cual puede usar. Si no tengo, el pedidos de pendientes en una cola, tengo un pedido cliente pendiente de tal cliente, los voy encolando.*

si tengo pedidos pendientes, lo saco y ahí le puedo contestar al lciente con la unidad que puede utilizar, el wait, poner al proceso que esta espradno y signal, sacar del proceso pop.

Alumnos mandan send a través del cnala atencion, mensaje, soy el proceso tal y van a recibir su rta por su canal privado, consulta a ese docente y respond a travees de su canal privado, soy el docente x, si no es final resolver la rta y mandarle a ese alumno.

El alumno una vez que termino la interaccion con el doc, similar a cuando tegno un conjunto de posibilidades pero cuadno obtengo una rta sigo con esa hasta terminar.

**PAres intecractuantes**

El problema uqe quiero revolver, n proceso dato local que llamo b, y el objetivo es saber cual es el menor y cual es el mayor de estos, todos conzocan el valor mas chico y cual mas grande, a travs de info local info global,

arquitect, a una soluc centralizada o simetrica, todos conectados, arqui te anillo circular soluc en forma de anillo.

que pasa si lo que tegno es una soluc, donde es centralizada. A partir de qe todos los proc conzocan su valor, todos conozcan el max y el min, cuando ya los conoce comunica a los procesos, p1, p2, si tengo n procesos y n es central, una vez que el centra calcula cual es el max y el min, devuelve los valores, 2\*n-1 mensajes. Dos tipos dif de procesos, se van a quedar esperando cual es el min y cual es el max, parecido a clientes y servidores, neceisto que todos los procesos hayan mandado los valores, espera a que eseten todos y manda. Se podria usar u nunico canal de resultados?? Depende suponiendo que todos los procesos solo tomen un valor, pero ademas si uno tuviera una primitiva que le permite mandar el mismo mensaje a todos los procesos, pero en el caso gral 2n-1 mensaje.

Como soluc es simple, pero al menos dos desventajas, si agrego procesos clietnes, los mensajes y tiempo de ejec crece linealmente, si hubiera algun problema donde el nodo central se cae todo se cae.

Que ocurre si tengo una arqu donde tods estan comun con todos, c u le manda su valor a otros, entonces cada uno recibe el valr de todos los otros y podria calcular el max y el min.

Todos iguales, enviar a todos los demas, a través del canal valores, su valor, desp pueden recibir los demas, y va haciendo min o max, en paralelo todal a arqu esta calculando, n\*n-1

conceptualmente tengo una solución SPMD, el mismo programa sobre datos distintos, dos etapas, uno llena los canales de valores y luego desagoto esos canales para ir sacando el max y el min, el nro de mensajes crece cuadráticamente,

**Pares interactuantes--anillo**

tendría que pasar all p0 el p1, p1 recibe y se fija, calcula, al p2, se fija los valores, hasta q la info de max y min llega, el proceso cero tiene que informar a los demás pero solo el uqe inicio la vuelta conoce la info completa, comunica el p1, el p2 al p3 y asi, una soluc en 2 etapas, calcula el max y el min y en la segunda si no recibiera durante determinado tiempo una rta tomar alguna acción.

esta soluc en forma de anillo es

Ventajas mensajes asinc, dejan mensajes lo dejen y no necesita n esperar a que sean tomados por otros.

Cleitne le pide al servidor tiene q esperar, clietne servidor, son **interac bidirecc y sicnrnismo,** asincronico no me da tantas ventanas. Remoto, no necesito de ida y vuelta.

Filtro y pares sin más ventajoso.

**Mensajes sincronicos**

Los canales son de tipo link o punto a punto (1 emisor y 1 receptor).

La diferencia entre PMA y PMS es la primitiva de trasmisión Send. En **PMS es bloqueante y la llamaremos** (por ahora)

**sync\_send.** Tienen que sincronizar en cada sincronización.

• El trasmisor queda esperando que el mensaje sea recibido por el receptor.

• La cola de mensajes asociada con un send sobre un canal se reduce a 1 mensaje ⇒ menos memoria.

• Naturalmente el grado de concurrencia se reduce respecto de la sincronización por PMA (los emisores se bloquean).  
- Los canales tienen capacidad de un mensaje

La primitiva de send y sinc send, la semantica es distinta y la probabilidad de deadlock es meyor.

interaccion de tipo prod, consumidor, valores es el canal, productor itera de a un, y son tomados, si tuviera dif entre velocidades entre productores y consumidores, si uso sincronicos send recive, se van completar teninendo en cunata cual es el uqe mas tarda, si uso asincronicos, si el productor es mas rapido, pero si es consumidor mas rapido va consumiendo de la cola de mensajes. (mayor concurrencia)

Para poder hacer esto mismo con los mensajes sincronicos, por ejemplo, tendría un proceso se encargue de que no se noten las dif de velocidad, recibe desde el productor y le envia al consumidor, eliminar las esperas entre el prod y el consu, el grado de concu se reduce cuando usa mensajes   
sincro en cliente servidor, cuando el cliente libera recurso tendria que poder dejar el recurso y siga, pero debe demorarse hasta que sea tomado por el sevidor, l**a posib de quedar en deadlock, es mas alta en mensajes sincronicos,**  
  
si ambos envian y desp reciben tienen que esperar lo mismo ambos entonces se produciria deadlock. con mensjaes sincronicos deberia p1 enviar y desp recibir, y p2 viseversa. Si fueran asincronicos si podria usar la solucion del lado izquierdo.

10. **Pasaje de mensajes con espera selectiva, comunicacion guardada.**

ccp link punto a punto, unidireccionales, nombrados, envio y recibo mensajes con sentencias de entrada ? y de salida!, envio y recepcion de mensajes.

process A { . . . B ! e; . . . } process B { . . . A ? x; . . . }

para que se produzca la intecraccion debe matchear, mismo canal y a tal proceso, es como tener una asignación distribuida.

Destino ! port(e1, ..., en); hacia donde por uqe canal y que variables  
Fuente ? port(x1, ..., xn); y desde donde

Port me permite distinguir entre distintas clases de mensajes.

procesos únicos o un arreglo, cuando recibo fuente.

Destino y Fuente nombran un proceso simple, o un elemento de un

arreglo de procesos. Fuente puede nombrar cualquier elemento de un

arreglo (Fuente[\*]).

• port son etiquetas que se usan para distinguir entre distintas clases de

mensajes que un proceso podría recibir (puede omitirse si es sólo

uno).

• Dos procesos se comunican cuando ejecutan sentencias de

comunicación que hacen matching.

A ! canaluno(dato); B ? canaluno(resultado);

copiar y oeste tienen uqe sincronizar en cada una.

interaccion de tipo productor y consumidor.

Cleinte servidor, el servidor a priori no conoce cual es el cliente q le va a pedir algo, mcd, tratar de recibir desde caulq de los clientes, por el canal arqs, recibe quien es el cliene y calcula el max como un div, responde por el canal resultado al clietne.

Process MCD

{ int id, x, y;

do true →

Cliente[\*] ? args(id, x, y); //recibe de cualqueir de los clientes

do x > y → x := x - y;

x < y → y := y - x;

od

Cliente[id] ! resultado(x);//envia de esta manera

od

}

*Comunicacion guardada, el hecho de tenr e / s bloqeuantes, hace que la concurrencia baje, deadlock suba,* si hay un proceso que tiene q comunicarse con otros por distintas entradas.

De quien quiero recibir? de un arreglo de procesos pero no de distinto tipo, copiar podría extenderlo, y en vez de un valor varios, buffering de un conjunto de caracteres, recibir otro o sacar uno de los que tiene, en un momento determinado hacer dif cosas dependiendo de quien este listo para comunicarse.

Espera hasta que una determinada cond sea verdadera, B; C → S;

B condicions//si no esta es verd

c sentenci de comunicacion //b y c la guarda

s conj de sentencias   
  
• La guarda tiene éxito si B es true y ejecutar C no causa demora.

• La guarda falla si B es falsa.

• La guarda se bloquea si B es true pero C no puede ejecutarse

inmediatamente.  
  
Envio o recibo algo que aún no está listo.

Como influye esto en la eje de las sentencias de tipo if o de tipo do?

id no deterministica, evaluo las guardas, si todas fallan el if termina. si al menos una tiene exito, se elige para ejecutar, si alguans se bloqeuan se espera hasta uqe alguna tenga exito,. se ejecuta la sentencia de la guarda y desp la sentencia s,

como las sentencias if , aparecen estas.

Do, la evaluacion de las condiciones se realizan en cada evaluacion del do,   
  
El de los caracteres, estructuras do tratando de recibir desde el proceso o este enviando al este.   
  
do donde tengo una sentencia de comnic guardada, tratar de recibir desde oeste, la comunicación no puede falllar, o tiene exito cuadno oeste manda un mensaje o no, una vez que recibi este caracter es sacarlo cuando recibi una sentencia de comunicacion.

q nos permite pensarlo asi? proceso copiar para manejar un buffer de tamaño 2, cuando me llego un caracter.  
  
copiar con buffer limitado.  
Process Copiar

{ char buffer[80];

int front = 0, rear = 0, cantidad = 0;

do cantidad <80; Oeste?(buffer[rear]) → cantidad = cantidad + 1;

rear = (rear + 1) MOD 80;

cantidad >0; Este!(buffer[front]) → cantidad := cantidad - 1;

front := (front + 1) MOD 80;

od

}

Si en un momento tengo elementos y no está lleno, podria recibir un nuevo carac o sacar alguno,, sino no. Si tengo menos de 80 y me quieren enviar lo puedo aceptas, cuando este vacio el unico que puede tener exito, es vacio sentencia <80 exitos, lleno cantidad >0 ese exito,

depende que pase con los procesos que quieran comunicarse conmigo

sincronico copiar es el buffering de tamaño 80, copiar me hace de amortiguador de las dif de velocidades de oeste y este. pueden aparecer cuadno necesito que no se note la dif de velocidades entre uno y el otro. Me permite resolver estos problemas de product y consumidores.

**Cliente servidor-** alocacion dee rec, uno le pide y otro da el recurso, y lo libera para q sea usado el recurso.

Process Alocador(servidor)

{ int disponible = MaxUnidades;

set unidades = valores iniciales;

int indice, idUnidad;

do disponible >0; cliente[\*] ? acquire(indice) → disponible = disponible -1;

remove (unidades, idUnidad);

cliente[indice] ! reply(idUnidad);

cliente[\*] ? release(indice, idUnidad) → disponible = disponible +1;

insert (unidades, idUnidad);

od

}  
  
La solución es concisa. Usa múltiples ports y un brazo del do para atender cada una. Se demora en un mensaje acquire hasta que haya unidades, y no es necesario salvar los pedidos pendientes  
  
cuando tenga unidades disp, y algun clietne que me pida la unidad, acquire, darle una unidad a ese proceso, saco una y le contesto a ese enviando un mesnaje diciendo que unidad es la que puedo usar O incondic, puedo recibir a través del canal realice, de cualqueir de los procesos que mensaje liberar, el proceso con tal indice me dice cual puedo usar.

Qué diferencias aparecen respecto al mensaje asincrónico?

Desde el lado del servidor, va a buscar un mensaje, si el proceso que lo habia mandado queria tomar una unidad del recurso, si tenia la unidad se la da, sino guardaba el pedido en una cola e ir a buscar otro hasta uqe apareciera devolver una unidad de rec, se fijaba is habia pedidos pendientes, lo sacaba de la cola y lo devolvia.

Acá si no satiface, lo guardo y voy a buscar otro, no lo voy a recibir si no tengo unidades disp, solo cuando tengo unidades disp recibo sino ni recibo. Solo mensajes de liebrar una undiad, implicitamente va a estar demorado porqe su pedido no va a ser tomado hasta uqe haya posibilidad de satisfacerlo, el clietne no sbae por que no fue tomado, pone el mensaje en el canal y queda demorado hasta qe sea tomado, no necesito guardarme los pedios pendientes, hasta uqe haya algun recurso que tenga los pedidos disponibles.   
  
recibe de cualqueira de los clietnes, si hubiera mas de uno, a priori no se de cual es.

Process P1

{ int valor1 = 1, valor2;

if P2 ! (valor1) → P2 ? (valor2);

P2 ? (valor2) → P2 ! (valor1);

fi

}

Process P2

{ int valor1 , valor2 = 2;

if P1 ! (valor2) → P1 ? (valor1);

P1 ? (valor1) → P1 ! (valor2);

fi

}

Esta solución simétrica NO tiene deadlock porque el no determinismo en ambos procesos hace que se acoplen las comunicaciones correctamente. Si bien es simétrica, es más compleja que la de PMA.  
  
no tiene deadlock porque es no deterministico, y si bien es simetrica es mas compleja que la de menjsaes asincronicros.   
  
puedo paralilizar el ejercicio de los primos con un proceso del tipo filtro, la salida de uno sea la entrada del que sigue, pasarle al sucesor los que no son multiplos de el, voy a tener un primer proceso, quedarse con el primer numero y desp itera enviando 3 a n con paso dos , con los impares, enviando , recibir un primer valor ese valor es un numero primo no fue multipl ode ninguno de los anteriores, empezar a recibir desde su antecesor, y si no es divisible, mientras analizo uqe pasa en uno le estoy pasando al siguietne,

En este algoritmo, excepto el primero uqe itera entre tres, como están escritos el resto, todos van a quedar bloqueados esperando el del predecesor, puede resolverse si mando un valor centinela ultimo, como cuando mandábamos zzz. Para modificar CRIBA.   
  
Ordenacion de un arreglo.

cuadno los valores qeu tienen que intercambiar ya no les sirven ya no los cambian y termina.

como las sentencias de e y s son bloqueantes es asimétrico, tambien podria usar comunicacion guardada y tener una asimetrica.

desde las iteraciones en el mejor caso vna a intercambiar el primer par de valores se da cuenta uqe esta ordenado y deja de hacerlo.

Como intercambio de a un valor n sobre dos y una ronda mas,

Si quisiera usar mas de dos procesos, para ordenar en cada ronda los procesos intercambian con una pareja, 1 con 2, 23 34 45 51 , c u con c u, nueva iteración e intercambio los mismos qeda igual , deberia muevo uno y vuelvo , en las rondas impares p1, c proc con impar, hace p1 y cada proce par p2, y viseversa.

en cada ronda voy avanzando hacia un arreglo mas ordenado que el anterior, como hace cada proc para darse cuenta que tiene que terminar? suele darse como problema en programas distribuidos… en general conocen parte de la info no todo.. los últimos dos procesos pasaron por todas las rondas y no tuvieron cambio, a dif de lo que pasaba, cuando el valor que intercambie no es menor las rondas como pueden hacer para detectar que toda la lista está ordenada? no se puede detectar, conoce solo la suya y la del compañero, o tengo un proceso coordinador, desp de cada ronda le dice si alguno hizo cambio, me agrega la cant de mensajes de c ronda o c proceso ejecuta una cant suf de rondas para garantizar que todo está ordenado, v procesos v rondas,   
Mensajes sincronicos como asinconicos, unidireccionales, productor ocnsumidor o filtro y rtb a pares que interactuan, porque la comunicacion que se da es unidireccional, interaccion entre pares trabajan indep y en algun momento comparten.   
VER CLASE 9.

Clase 10.

Programas distribuidos, se comunican solo por mensaje.

En particular los problemas de tipo clietne serv que son bidirecionales, mejor resolverlo con mecanismos bidi, mecanismos de comuni y sinc, y asumen que lo que hay es bidireccional, las uqe vamos a ver son asi. RPC y Rendezvous, combinar una interfaz, los procesos las invocan a través de llamadas de tipo call, esto lo combinan con mensajes sincronicos, el proc q envia el pedido, que hace el call, lo que hace es al proceso demorar al llamador hasta que la op que invoco es atendida se termina de ejecutar y se devuelve el resultado, ese call, envía un pedido, demora al que hace el pedido, hasta que esa operación es terminada de atender y se el da un resultado.

RPC   
Dif en la manera que tienen de atender los pedidos de operación.

RPC, tener un procedimiento diferente para cada una de las op que estoy brindando como servidor, y se crea un nuevo proceso para atender c u de esos llamados, clientes hacen llamados a esas operaciones y cada vez que hay un llamado se hace, el que hace el llamado y el que lo atiende pueden estar en diferentes máquinas, remoto, para el proc qe hace el cliente cuando hace el pedido no le importa dónde está.

Cada vez que hay un llamado, se genera un nuevo proceso.

Vent:grado de paralelismo es mas alto.

Desv: la implementacion de este mecanismo es mas compleja, que el otro enfoque.

Rendezvous

Un proceso que ya existe, y cuando hay un pedido es atendido por una sentencia de entrada, espera que haya una invocación, la procesa y da el resultado.

Ya existe el proceso que atiende de a uno.  
  
Un programa que usa RPC va a tener un conjunto de módulos, con procesos y procedimientos, que pueden estar en espacio de direc dif, máquinas dif, film 7, esquema general.

Los procesos de un módulo pueden compartir variables y llamar a procédures de ese módulo.

Un proceso en un módulo puede comunicarse con procesos de otro módulo sólo invocando procedimientos exportados por éste. No obtengo interacción directa de un proceso de un módulo y un proceso interno de otro módulo.

Los módulos tienen especificación e implementación de procedures

module Mname

headers de procedures exportados (visibles)

body

declaraciones de variables

código de inicialización

cuerpos de procedures exportados

procedures y procesos locales

end

Los procesos locales son llamados background para distinguirlos de las operaciones exportadas.

Header de un procedure visible:

op opname (formales)[returnsresult]

El cuerpo de un procedure visible es contenido en una declaración proc:

proc opname(identif. formales)   
 returns identificador resultado

declaración de variables locales

sentencias

end

Un proceso (o procedure) en un módulo llama a un procedure en otro ejecutando:

call Mname.opname (argumentos)

¿Qué ocurre cuando sucede un call?

Para un llamado local, el nombre del módulo se puede omitir.

El proceso llamador en algun momento invoca a un proceso que está en otro modulo, ese call va a producir que el módulo donde se exporta se genere/se crea un proceso servidor, que va a atender ese pedido, inicia la ejecución del proceso servidor y cuando termina devuelve los resultados y desaparece. No existe antes, ni después.   
RPC es mas uqe nada un mecanismo de comunicación, si bien sincronizan, el que invoca y el que sirve es implicita.   
La sincronización entre el que invoca y el que sirve es implícita, dentro de un módulo vamos a necesitar que los procesos que están en el → sincronicen, los procesos propios del módulo y lo que fueron levantando para atender el llamado que ellos sincronicen, en algun momento podria haber mas de un proceso dentro de un módulo, entonces puedo manejar la sincronización con exclusión mutua(uno solo a la vez) o si ejecutan concurrentemente dentro del módulo, em, las variables compartidas protegidas de forma automática, no voy a tener interferencia, si ejecutan concurrentemente necesito mecanismo para em y sinc por cond, cada módulo ahora un programa concurrente con memoria compartida, cada máquina, parte de mi programa constituido por un modulo, en una máquina y dentro de un modulo podria tener mas de un proceso ejecutando a la vez.   
  
12. Ejemplo.

brinda servicios de timing a los demas, dos op visibles,

cada vez que pasa un instante se fija si tiene uno o mas proce que despertar, gana el acceso exclusivo con el pm , saca y lo despierta con el v.

cuando termino de despertar todos op v sobre el semaforo m,

visualizar, que ante cada llamado de proc clietne al generarse un nuevo proceso para atenderlo, si estos procesos acceden a variables compartidas, necesito protegerlo es como un monitor no exclusivo, brindo op e invoco ops. es no exlisivo en un momento puedo tener mas de uno ejecutando, entonces necesito proteger las variables compartidas.

Vamos a tener que tenemos procesos de aplic y un fileserver, llaman a op de read y write para acceder a los datos, los archivos van a estar en el fileserver y los modulos FS va a manejar el acc a los bloques del disco. El modulo Fcache, mantiene en la cache los bloques recientemente leidos, cuando recibe un pedido por esas lecturas, si los tiene los devuelve sino va a buscar los datos, si tiene los datos en la cache los puede buscas aca sino tiene que salir a buscarlos al Fs, invocando a la op que corresponda dentro de FS, el modulo File cache, brinda ops read y write, implementadas dentro del modulo, si no los tiene a los datos los busca al modulo Fs , si el cache tiene lugar selecciona los bloques y va a buscar, sino va a tener que vaciar parte de la cache, hacer lugar y desp hacer lo que resta.

readblock si lsos datos no estan en cache los busca en disco, encola la lect y espera a que sea procesada, en el modulo ademas esta el proceso driver que maneja los pedidos de acceso al disco, encola el pedido. Desde el punto de vista cliente servidor, pide y pide, pero a su vez se convierte en un cliente para el modulo fileserver, hay modulos que a veces son clietnes y a la vez servidor.

C/S adecuado pensar en RPC, porque si no por c interacción hubiera tenido que tener un canal mensaje de ida y canales de resp, como es bidireccional hago el pedido y espero. funciona bien entre cliente servidor pero para cliente consumidor o pares, no. Dos procesos de modulos diferente que quieren intercambiar valores, con mensajes asincronicos, ponen el valor y c u lo toma, sincronicos, comunicacion guardada o bloqeu o y desbloqueo, ahora puedo hacerlo llamando a otro modulo, cada modulo exporta una op que le permita a otro proceso invocarla, esta interacción es la que es visible desde el exterior,

envia el valor con el proceso que está en el otro modulo y queda esperando a que termine de ejec la operacion, entre dos modulos es complejo en tipo RPC, estoy intentnado de resolver algo que es unidireccional con algo que es bidireccional.

Si bien puedo resolverlo con varios algunos mecanismos es con los otros.

Clietnes o servidor → bidireccional RPC o RENDEVOUS.

Java es un ejemplo del RPC, metodos remotos rmi, interfaz a lso encabezados , clase servidor y

• Una interfase que declara los headers para métodosremotos.

• Una clase server que implementa la interfase.

• Uno o más clientes que llaman a los métodosremotos.

El server y los clientes pueden residir en máquinas diferentes.

Rendevouz, a dif del otro, tengo una comb de sincr y comunicacion, los progs son conj de modulos, invoca con call, pero esta va a ser servida por un proceso que ya existe en lugar de por uno nuevo, va a usar una sentencia de entrada para actuar en funcion del llamado, operaciones se van a atender de a una, no concurrentemente, la especificacion del modulo va a contener los headers, pero un unico proceso que va a servir estas op, entra en rendevous, sincroniza con un llamador, con una sentencia de entrada, cuando el servidor llega a un lugar donde tiene una sentencia de este tipo, puede atender un pedido por una operacion, atiende elv pedido y luego el conjunto de acciones, operacion guardada,

in opname (parámetrosformales)→ S; ni

Las partes entre in y ni se llaman operación guardada.

una sentencia de entrada, se va a demorar hasta que haya al menos algun pedido con esta operación, se detiene hasta que haya algun pedido por esta operacion si hubiera agarr el mas viejo, copia los args de los params form, ejecuta y devuelve los params de resutlado al que hizo el llamado,

proceso clietne, llega un call, lo hace queda esperando a que termine, como se procesa?

en algun momento llega la op in, ejecuta las sentecnais y desp sigue ejecutndo. cuando llega un in, lo atiende si habia pedidos guardados.

puedo combinar la comunic guardada con rendevous, no deterministica, los cuadraditos, puedo atender por la op uno o por otra, podria elegir no deterministicamente por alguna de las operaciones.

que haces las politicas de shce? ordenado por e1, permitirem cambiar el orden de cola de pedidos, los pedidos podrian atenderse de acuerdo al parametro qu le mande,

in op1 (formales1) and B1 by e1 → S1;

... opn (formalesn) and Bn by en → Sn;

**IN, muy poderosa, me permite atender pedidos por distintas operacione,s atenderlos si se dan determ condic, y ademas por algo otro orden que no esa lifo o fifo. Pero siempre de a una operacion a la vez!!!**

no necesito proteger porque atiende de a uno!

Time server se ordenan de acuerdo a la hora!! el efecto es no necesito cola adicional para los pedidos, voy atendiendo en el momento y orden qu corresponde.   
  
SJN

op para atender y liberar, para no tenre dem innecesaria. liberar es atendida incondicionalmente, mientras que pedir el rec cuando esté libre   
 **ADA - Rendevouz**

**in = accept**

**entry los puntos de invocación**

exp de scheduling, es en forma de cola, y no puedo hacer ref en la atención a los parám con los que hice el llamado.

Heartbeat- Sit que pueden darse en la prog distribuida, tengo un conj de procesadores, canales bidirec, c u se conoce con su vecino y conocen los links q lo conectan con sus vecinos, como pueden c u , saber que procesador esta conectado con otro, los enlaces como canales compartidos, los procesos vecinos interactuan intercambiando conf total.

exp enviando info y contraerse recibiendo info, serie de etapas donde los proc mandan y reciben info, se expanden y se contraen los procesos, como puedo usar este problema con este algortirmo, procesadores como n proceso nodo y los vecinos arreglos, una red es un grafo, procesos conectados, si la red es un grafo, calcular la matriz de adyacencia, puedo hacer que cada nodo tenga la topologia completa, desp de un par de rondas voy a saber que pasa con toda la red, como en un grafo voy teniendo los niveles.

Clase 11-

paralelo es concurrente pero ejecutado sobre un procesador, dif maquinas dif procesadores, se busca mejorar el tiempo de ejecución de un programa, fundamentalmente

Hay diferentes enfoques para clasificar las arquitecturas

paralelas:

• Por la organización del espacio de direcciones.--> si comparten memoria o no

• Por la granularidad.

• Por el mecanismo de control. → como se ejecutan las instrucciones.

• Por la red de interconexión. → como estan conectados los procesadores

Las arquitecturas paralelas se clasifican según su espacio de direcciones en:

• Memoria Compartida. →

• Memoria Distribuida.  
mem compartida o pasaje de mensajes

Esta clasificación se relaciona con el modelo de comunicación a

utilizar:

• Accesos a Memoria Compartida (memoria compartida). → la interacción se da modificando datos en esta mem compartida, direfenciancion entre esquemas uma y numa, mas de un procesador, uma acc uniforme a mem, memoria desde cualquiera de los procesadores me cuesta el mismo tiempo acceder. numa→ c u su propia memoria, acceso no uniforme.

Problema basico: en ambos casos mantener consistente los valores de las cache y las memorias.

Ejemplo de multiprocesador de memoria compartida: GPU. maquina de grano muy fino.

multiproc con mem dist, una red de interconecion y pares de mem y comunic conectados, interaccion a trvés del pasaje de mensajes.

cluster de grano gureso, mucha capacidad de cpmputo y la red no es tan optimizada.

• Intercambio de mensajes(principalmente memoria distribuida).

En algunos casos también tenemos en la misma plataforma ambos

mecanismos.

Por mecanismo de control.

Propuesta por Flynn (“Some computer organizations and their effectiveness”, 1972).

Se basa en la manera en que las instrucciones son ejecutadas sobre

los datos.

Clasifica las arquitecturas en 4 clases:

• SISD (Single Instruction Single Data).

• SIMD (Single Instruction Multiple Data).

• MIMD (Multiple Instruction Multiple Data).

• SISD(una inst sobre un dato) monoproc, una unidad de procesamiento, una unidad de control y una memoria desde la cual la mem toma los datos que necesita, la unidad de control tb recibe desde la memoria. Asoc tb a la ejecución secuencial y deterministica.

• Instrucciones ejecutadas en secuencia, una por ciclo de instrucción.

• La memoria afectada es usada sólo por ésta instrucción.

• Usada por la mayoría de los uní procesadores.

• La CPU ejecuta instrucciones (decodificadas por la UC) sobre los datos.

La memoria recibe y almacena datos en las escrituras, y brinda datos en

las lecturas.

• Ejecución determinística.

• SIMD (Single Instruction Multiple Data).

flujo de de instr dif pero sobre el mismo dato, varios procesadores que reciben desde las unidades de control, las instrucc y desde la memoria reciben un dato, multiples algoritmos tratando de descifrar algo encriptado.   
• Conjunto de procesadores idénticos, con sus memorias, que ejecutan la

misma instrucción sobre distintos datos.

• Los procesadores en general son muy simples.

• El host hace broadcast de la instrucción. Ejecución sincrónica y

determinística.

• Pueden deshabilitarse y habilitarse selectivamente procesadores para que

ejecuten o no instrucciones.

• Adecuados para aplicaciones con alto grado de regularidad, (por ejemplo

procesamiento de imágenes).

• MISD (Multiple Instruction Single Data).

conj de proc iguales qeu en un instante de tiempo estan ejec la misma instr sobre distn datos, sincronica y deterministica, todos tardan lo mismo entonces luego estna listos a la vez. muy util cuando tenes apic con imagenes por ejem,

• MIMD (Multiple Instruction Multiple Data).

c proc tiene su prop flujo de isntr y de datos, c u a su ritmo, no es sincronico, c u su prop unidad de control, pueden o no tener mem compartida, y tienen una red de interconecciom  
  
• Por la red de interconexión.

mem comp como pm, puede contrs maq conectando proc y mem, redes estaticas, utliziadas en maq de mem dist, y dinamicas, compartidas swtiches y enlaces de comunic, ejemplos crossbar

buss acceso limitado, costobajo

**ALGORITMOS PARALELOS**

La mejor solución puede diferir totalmente de la sugerida por los algoritmos secuenciales existentes.

Puede darse un enfoque metódico para maximizar el rango de opciones consideradas, brindar mecanismos para evaluar las alternativas, y reducir el costo de backtracking por malas elecciones.

Aspectos independientes de la máquina tales como la concurrencia son considerados tempranamente, y los aspectos específicos de la máquina se demoran.

¿Por qué es compleja la programación paralela?

• Decidir cuál es la granularidad óptima de las tareas.

• Mapear tareas y datos a los nodos físicos de procesamiento (¿en forma

estática o dinámica?)

• Manejar comunicación y sincronización.

• Asegurar corrección. Evitar deadlocks. Evitar desbalances.

• Obtener un cierto grado de Tolerancia a Fallos.

• Manejar la heterogeneidad. → si tengo procesadores y tareas de distintos tipos, ver como lo manejo

• Lograr escalabilidad en todos los casos (potencia, tamaño de la arquitectura y

del problema). → ver si mi app crece con la maquina.

• Consumo energético.

Para diseñar un algoritmo paralelo se deben realizar alguno de los siguientes pasos:

• Identificar porciones de trabajo (tareas) concurrentes.

• Mapear tareas a procesos en distintos procesadores.

• Distribuir datos de entrada, intermedios y de salida.

• Manejo de acceso a datos compartidos.

• Sincronizar procesos.

Pasos Fundamentales: Descomposición en Tareas y Mapeo de Procesos a Procesadores.

en las primeras etapas, tomar el problema y encotnrar el mayor grado de concu posible, como se comunican e intentar agrupar en la aglomeración, porque puedo agrupar y tener dif conf posibles, si de entrada tengo muy pocas tareas en todo momento 4 tareas ejec al mismo tiempo, 256 tarea. podria tener mas tareas ejec al mismo tiempo.

Lo importante es que en el momento q me encontre con un conj de tareas y con un conj de procesadores es decir esta tarea la pongo en este proc, no existe un alg q me permita mapear tarea a los procesadores,

**Métricas del paralelismo**

En el mundo serial la performance con frecuencia es medida teniendo en cuenta los requerimientos de tiempo y memoria de un programa.

En un algoritmo paralelo para resolver un problema interesa saber cuál es la ganancia en performance.

Hay otras medidas que deben tenerse en cuenta siempre que favorezcan a sistemas con mejor tiempo de ejecución.

A falta de un modelo unificador de cómputo paralelo, el tiempo de ejecución depende del tamaño de la entrada y de la arquitectura y número de procesadores (sistema paralelo = algoritmo + arquitectura sobre la que se implementa)

La diversidad torna complejo el análisis de performance...

• ¿Qué interesa medir?

• ¿Qué indica que un sistema paralelo es mejor que otro?

• ¿Qué sucede si agrego procesadores?

En la medición de performance es usual elegir un problema y testear el tiempo variando el número de procesadores. Aquí subyacen las nociones de speedup y eficiencia, y la ley de Amdahl.

Otro tema de interés es la escalabilidad, que da una medida de usar

eficientemente un número creciente de procesadores.

medida que me permite analizar solución paralela,

S es el cociente entre el tiempo de ejecución del algoritmo serial

conocido más rápido (Ts) porque siempre la comparacion la hago contra la mejor solucion que haya, y el tiempo de ejecución paralelo del algoritmo elegido (Tp) :

va a estar entre 0 y en el caso de arq homog P. si yo utlizo 8 proc, para resolver una app en forma paralela busco hacerlo 8 veces mas rapido, entocnes en gral, este speedup suele graficarse con una curva, cant de procesadores y speed up, el optimo es q sea la curva identidad, para una cant de procesadores el speeedup up no llega, cuatno mas rapido ejecuto una soluc en forma paralela.

Speedup superlineal, en la ejec paralela al utilizar mas de un proc, cada tanto bajadas a disco para poder manerar esas ED , o que mi apl paralela encuentre la solucion que lo uqe la hacia en forma secuencial, si paralelizo y le doy una rama de mi arbol a c procesador, relacionado con el trabajo total que voy a realizar del algoritmo.

EFICIENCIA - otra medida

cociente entre speed up y speedup optimo

Mide la fracción de tiempo en que los procesadores son útiles para el cómputo.

El valor está entre 0 y 1, dependiendo de la efectividad de uso de los procesadores. Cuando es 1 corresponde al speedup perfecto.

fracción de tiempo que son optimos para procesadores.

Escalabilidad de los Sistemas Paralelos

capaz me sirve para pocos, pero para varios no.

la capac q tiene un sistema, de mantener la efic constante cuando crece la canti de procesadores y tb los datos donde estoy procesando.

si ejecuta bien, con pocos procesadores y datos, tb deberia tenerla con muchos procesadoresy muchos datos.

Factores que limitan el speedup

- un porcentaje alto de codigo secuencual, ley de amdahi

-puede haber alto porc de e/s y si me la paso yendo a disco, tb cae

-el algoritmo no sea adecuado

-excesiva contencion de mem, mismo tiempo procesos acediendo a los mismos datos.

-tamañ del problem, regla general es que el paralelismo sirve para problemas grandes o que sea fijo y yo quier usar mas proc.

- Desbalance de carga (produciendo esperas ociosas en algunos procesadores).

- Overhead paralelo: ciclos adicionales de CPU para crear procesos,sincronizar.

amahhdi: para un problema dado, existe un maximo alcanzable indep del num de procesadores.

si crezco la cant de proc tb trato de crecer el tam de mi problema. porque la parte paralelizable crezca.

Paradigma de programación: clase de algoritmos que resuelve distintos problemas, pero tienen la misma estructura de control. En cada paradigma los patrones son iguales.

Para cada paradigma puede escribirse un esqueleto algorítmico que define la estructura de control común.

Dentro de la programación paralela pueden encontrarse paradigmas que permiten encuadrar los problemas en alguno de ellos.

En cada paradigma, los patrones de comunicación son muy similares en todoslos casos.

Basado en organizaciones del mundo real.

el maestro le dice al esclavo que hacer, el esclavo cuando termina le da los datos, tareas a realizar a c u. esquema sencillo pero si la cant de esclavos crece todos quieren acceder al master, resolver cual es el tam de la tarea, si uso tarea muy chicas todo el tiempo mandando y recibiendo results, el cuello de botella, si tengo tareas muy grandes la soluc paralela termina cuando va a terminar el último de los procesos.

• El master envía iterativamente datos a los workers y recibe resultados de éstos.

• Posible “cuello de botella” (por ejemplo, por tareas muy chicas o slaves muy

rápidos) → elección del grano adecuado.

• Dos casos de acuerdo a las dependencias de las iteraciones:

Iteraciones dependientes: el master necesita los resultados de todos los workers

para generar un nuevo conjunto de datos.

Entradas de datos independientes: los datos llegan al maestro, que no necesita

resultados anteriores para generar un nuevo conjunto de datos

Pipeline → secuecializado  
• El problema se particiona en una secuencia de pasos. El stream de datos pasa entre los procesos, y cada uno realiza una tarea sobre él.

• Ejemplo: filtrado, etiquetado y análisis de escena en imágenes.

• Mapeo natural a un arreglo lineal de procesadores.

• Extensiones:

Procesadores especializados no iguales.

Más de un procesador para una tarea determinada.

El flujo puede no ser una línea simple (ejemplo: ensamble de autos con varias líneas que son combinadas) → procesamiento sistólico.

cuadno termino la etapa uno y arracna la dos ,a uno ya puede empezar a ejec otra etapa.

ganancia si tengo un prob de 4 etapas, 8 unid de tiempo, mietnras que si uso pipe muchas menos.

**Dividir y conquistar**, paralelismo recursivo, problema general descomponerlo en partes rec que trbajan sobre el problema total,

tengo un problema , si no se al mas chico al mas chico, que se como se resuelve y empiezo a resolver para atras.

en lugar

muchos problemas recursivos puede ejec de forma paralela.  
  
SPMD

el prog genera un unico prog que ejec en un unico programa sobre una porcion del dominio de datos, c nodo distintos caminos del progrmaa, como genero el programa y como distribuyo los datos?

El programador genera un programa único que ejecuta cada nodo sobre una

porción del dominio de datos. La diferente evaluación de un predicado en

sentencias condicionales permite que cada nodo tome distintos caminos del

programa

Dos fases: 1) elección de la distribución de datos y 2) generación del SPMD

1) Determina el lugar que ocuparán los datos

en los nodos. La carga es proporcional al

número de datos asignado a cada nodo.

Dificultades en computaciones irregulares y

máquinas heterogéneas.

2) Convierte al programa secuencial en SPMD.

En la mayoría de los lenguajes, depende de

la distribución de datos.

Suele implicar paralelismo iterativo donde un programa consta de un conjunto de procesos los cuales tiene 1 o más loops. Cada proceso es un

programa iterativo.

Generalmente, el dominio de datos se divide entre los procesos siguiendo diferentes patrones.