**Concurrencia**

**¿Qué es?**

La capacidad de realizar múltiples actividades al mismo tiempo. Desde el Hw hasta cómo están hechas, importante en los SO que nos permitan usar concurrencia en los programas que hagamos.

No **determinismo**, en ejecución concurrentes con la misma entrada se pueden obtener resultados diferentes *depende del orden en que se van ejecutando las cosas*.

Mismo programa, mismas entradas pero por la forma en que se van dando las cosas podría ser distinto el resultado, depende de la semántica del programa, si hay algo que no debería permitirse se resolvería con algo para eso.

Algunas **ventajas de la programacion concurrente**

- La velocidad de ejecución que se puede alcanzar.

- Mejor utilización de la CPU de cada procesador, de los cores.

- Explotación de la concurrencia inherente a la mayoría de los problemas reales, soluciones más sencilla de pensar/ver.

**Posibles comportamientos que van a tener los componentes de un programa concurrente.**

Un Programa concurrente va a estar formado por varios procesos, este programa es para resolver un único problema, ese programa va a estar compuesto por varios procesos, distintos comportamientos, pueden ser independientes los procesos cada uno resuelve su parte sin interactuar, pero vemos cuando estos procesos cooperan o compiten, entonces los procesos puede ser de tres formas, **independientes, cooperar o competir**

**Cooperar** c/u ejecuta una parte y después de alguna manera esas partes se integran para obtener el resultado final.

**Compiten** pelean para utilizar cierto recurso compartido(venta de pasaje de avion, usuarios compiten para acc a cierto pasaje)

Problemas:

**DeadLock** → agarre el ovillo y no lo suelta, cuando necesita hacer el otro no dejaron el anterior, ninguna puede continuar, van a estar esperando que el otro proceso libere un cierto recurso compartido que no lo deja hasta que no logre obtener el que necesita, entonces ninguno suelta su recurso hasta no obtener el otro y se produce deadlock(punto muerto), abrazo mortal porque ninguno proceso va a poder continuar.

**Inanición →** alguno de los procesos nunca logre conseguir el recurso porque el otro gana, cuando más procesos hay es más probable que esto se dé.

**Cooperación →** dividimos el problema en partes y luego lo unimos para formar lo que sea el resultado. Hay muchas decisiones que tomar, usamos 4 , o 5 tres que procesen y una que una? el que terminó antes? el más rápido? Lo mejor posible en cuanto al tiempo de ejecución.

Los procesos se comportan de las tres maneras, pueden intercalarse entre ellas.   
  
Trabajo paralelo, distribuir el trabajo de forma adeacuada

sin paralelismo sobre una sola máquina, reducir el tpo final de ejec no tanto pero sí menos, concurrencia es todo con paralelismo de hw o no, varias compus o no.

Distribuc de la carga de trabajo, como la maquina es una sola. a lo sumo tardon e nel context switch.

La necesidad de compartir recursos es menor.   
  
Mem compartida, dejando la info en alguna pos de mem y otro proc puede tomar el dato que necesita, todos los procesos deben acceder al mismo ED.

todos los proc deben acc al mismo ed.

Necesito alguna herram para bloquear y liberar el acceso a la pos de mem, para uqe no haya dos a la vez.

como x pas de mensaj, unidades de proc separadas entonces las comunicac a través de una red de interconexión, puede ser utilizado simulan un ed, pero los procesos no pueden acceder al ed de otro, maquinas arqui que tenga memoria compartida

Pasaje de mensajes. canales logicos o fisicos, pensado en arqui distribuidad, los procesos podrian estar en un multicore o monoproc, canal logico.

sincronizacion, y comunicacion cuales son los procesos y como se comunican y sinc.

Los proc se pueden sinc de dos maneras, por excl mutua o sinc por condicion, sirven para distintos obj, nosp permite asegurar q un unico proceso esta accediendo, y hasta q no termine nadie debería acceder.  
  
A lo sumo un proc a la vez!! Exc Mutua dentro de la seccion critica!

por condicion, bloquear hasta que determinada condicion ocurra, si necesito sacar un elem debo esperar para sacarlo.

problema ocnuc que usa ambos mecanismos, mem compartida, buffer al cual todos acceden, proc consumidores y proc produc, buffer limitado

toman o consumen hasta qe les avisan, ambos tipos de mecanismos, condicion y bloqueo, tambien sic por e.m si hay mas de un prod q quiere dejar un elemento. Van a tener que sincronizarse entre ellos por em para q lo tome un solo proceso.

Interferencia, supuso cierto estado, actual del prog y cuadno va a trabajar hay algo que modifico ese estado y puede provocar error,

un proceso que tiene mayor prioridad es decir q puede obligar a otro para q el de mayor prioridad acceda, como nos vamos a manejar en la practica. no le quita el recurso sino que lo puedo ceder

Programa poquito computo y comunicar, **granularidad fina**

Grano grueso, mucho codigo con menos cantidad de comunicacion.

multicore grano fino donde puedo div el proble en pequeños.

Manejos de recursos compartidos, admin el uso cuanod varios qieren usarlo, tenemos q admin su uso, con pc, la asginacion de recursos a los procesos, manejar metodos de acceso, bloqueo y liberacion de los recursos, cuadno libero tengo q decirle a tal q pase, SEGURIDAD.

**Fairness,** si todos tienen q usar un rec compartido que sea con las mismas posibilidades de ir accediendo,

Evitar la **INANICION,** cuando uno nunca logra usar un rec compartido.  
   
**Overloading**, sobrecarga de un proceso, mucho mas trabajao del que puede resolver

**Deadlock** dos o mas no puedan seguir con su ejecucion, evitar que queden sin trabajar digamos. Por error de programación ambos quedan esperand oque otro libere, un recurso compartido. La ausencia de deadlock es una propiedad necesaria en los programas concurrentes.   
  
Propiedad algo que tiene que producirse en todas las ejecuciones

4 condiciones que se tienen que dar para que exista deadlock e 2 o mas proc, rec reusables serialmente, rec compartidos pueden ser usados por un rec a la vez y se deben acceder o usar con ex.mutua una vez q un proc termino odeberia liberarlo y lo usa otro recurso, a medida que se van liberando

Adquisicion incremental, los proc van adquidriendo los proc compr q necesitan y se los van quedando, va tomando los que nnecesita

No apropiacion, si un proc tiene un cierto rec compartido nadie lo puede forzar a dejarlo,

espera cíclica un ciclo donde un proceso necesita un rec compartido que necesita el siguiente proceso, cada uno tiene un proceso pero necesita el que tiene el otro, ninguno va a liberar su rec compartido, los va tomando de a uno y el que tome no lo suelta hasta q termina de usarlo, dadas estas condiciones nadei podria avanzar, c u un rec dif, nadie podria adquirir el rec que necesita,   
  
si no se dan los ciclos no hay problema, a lo sumo 4 que obtuvieron un rec

Evitemos la posibilidad de que haya deadlock,

Requerimientos del lenguaje q vayamos a utulizar, java o librerias , open mp

Idependientemente vamos a necesitar algo uqe nos permita especificar cuales son las porc de codigo q se pueden ejec de forma concu, cuales son los modulo que van a formar parte del programa, los procesos deben poder comunicarse, y sic por cond o em. todo eso debe brindarme el lenguaje.

**Problemas asoc con la pc**

**no son indep, suelen compartir recursos, necesitamos mecanismos, los procesos inciados en un**

los procesos inic dentro de un prog conc puede ser q en algun momento ya no existan, deadlock o termino de forma correcta, esto indica una mala dist de los recursos o tarea

La forma en q se van intercalando, resultados finales o intermedios diferentes qe hacen dificil la interpretacion y el debug, a veces no logramos repetir ese errror como varía.

Posible reduc de performance, context switch tenemos q sacar un proz y darselo a otro, guarda el estado para ucando vuelva poder continuar, esto genera tiempo y overhead, retardo, si hago sinc pro cond, pasaje de mensajes la comunicacion tiene cierto costo.

Leva un mayor tiempo de desarrollo y puesta a punto, lleva tiempo tratar de lograr el mayor rendimiento posible, aprovechar lo mayor posible las carac de hw sobre el q trabajo muchas veces es dificil, paralelizar algoritmos secuenciales

procesos, como se comunican entre ellos y sincro, desp eso se ejecutara en un hw, arq paralela o monoproc.

programa en si ref concu a concu en gral

paralelismo, una parte dentro de la concurrencia, ejecutarlo en una arqui q tiene mas de una unidad de procesamiento, tratando de usar lo mejor posible la arq completa,   
  
**Clase 3**

Niveles de memoria sobretodo en memoria compartida.

Cuando mas cerca de los reg de la cpu el tpo de acce es mayor pero reduce el tamaño, aprovechar los niveles de memoria para sacar el mayor jugo posible.

Localidad espacial y temporal,

**Clases de instrucciones**

**Asignación**

asignacion simple x = e

Sentencia de asignacion compuesta x:=x+1 ; y:=y-1; a[2]=6 , aa[2,6]= a[4]  
Lamamado a funciones x=f(y)+g(6)-7

swap: v1:=:v2 → intercambiar los valores de dos variables

skip: termina inmediatamente y no tiene efecto sobre ninguna variable de programa. → si utilizamos, no hace nada no utiliza procesador no hace cambios.

Estruct de contro de **Alternativa**

If

if else

sentencia alternativas múltiples

no es como case, el case va en orden,

este if evalúa todas, las V elige una de forma aleatoria y ejecuta

si todas son falsas sale del if sin hacer nada mas.

**Iteración**

**For all**

como for común, más de un cuantificador,

itera por todas las combinaciones posibles

St para que esa st, se cumpla

Para esa iter no se ejecuta nada y se pasa directamente

while(cond) S

for [i=1 to no]

vuelve a iterar y chequea las condiciones hasta que encuentra que todas son falsas.

Cómo definir los pedacitos de código que se van a ejecutar de forma concurrente, dos opciones, mem compartida o pas mensajes,

definir con las sentencias **co**, único hilo hasta que en algún momento de necesita ejecutar algo de forma concurrente, cuando termina un único hilo, donde cada hilo ejecuta un pedacito de codigo, desp un unico hilo y termina. Dos maneras de hacer los llamados, si cada hilo un código dif, llamado a c u, si todos tuvieran que hacer podes poner un cuantificador y un código asoc a ese cuantificador, el codigo dentro c hilo dentro de las llaves, el primero de los procesos i valor ,1 valor 2 y asi ejecutando de forma concurrente, asi trabajan los lenguajes con mem compartida, un hilo y cuando se necesita varios hilos.

**Process** cada uno define los distintos procesos que van a formar parte del prog concu y van a empezar a correr todos juntos desde el inicio del programa, todos la misma prioridad y empiezan a ejecutarse, y establecer el conj de sentencia que debe ejecutar el proceso.

Process B todos procesos iguales, no necesitamos definir un codigo para cada uno. N copias del proceso B, y con el conj de sentencias q deben haber,

los Process ejecutan en background de entrada siempre, la sentencia co

un unico hilo y cuando es necesario recine para ejec algo en forma concu.   
 **CLASE 4**

Estado valor que tienen las variables, los registros en un cierto momento y de alguna manera guarda los valores esos serian el estado en un programa concu, cuando un proc tiene q ejec un pc, eje sentencias, cada sentencia formada por una o mas acc atom mientr

intercalado de las acciones atom, entre los dif procesos, la secuencia de estados por los q se pasa, forman una historia no todas son validas, secu validad y otras no,

estados intermedios que se pueden llevar a cabo mientras ocurre la acc no puede llevarse a cabo otro, c prog su acci atom,

hay historias validas y otras no,

**La sinc por condición permite restringir las historias de un programa concurrente para asegurar el orden temporal necesario.**

**Grano fino,** no deben generar nada opr sw , grano grueso si sinc por em para uqe se del atomicamente, una op de asginac no es atomica porqe internamente eso involucra dos acciones**,**

**grano fino, los valores se leen o escribenn como acc atomicas,** una const o alguna variable simple van a ser acciones atomicas de grano fino,   
  
Los valores se cargan en registros, se opera sobre ellos, y luego se almacenan los resultados en memoria.

Cada proceso tiene su propio conj de registros(context switch)

Todo resultado intermedio se almacena en reg o memoria privada del proceso.

**Referencia crítica en una expresión** ⇒ referencia a una variable que es modificada por otro proceso. Asumamos que toda referencia crítica es a una variable simple leída y escrita atómicamente

A lo sumo una vez si a lo sumo una ref critica y x no es rerf por otro proceso

1. ref a una v comp q es modfi por otro proc,
2. e no cont ref criticas, en cuyo caso puede ser leida.

A lo sumo una variable compartida y referenciada una vez.

Si una sentencia cumple la prop su ejec parece atom aunque esto involucre acciones atom de grano fino y haga ref a una var compartida de otro proceso.

a ver si cumple la propp

unica expresion

dos variables comp, x e y, uno inc x, otro inc y, ambas son compartidas se las trabaja como locales. no hay ref criticas ambas sentencias cunplen las prop de a lo sumo una vez, un único proc hace ref a x y uno solo a y.

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

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

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

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

Nunca debería ocurrir x = 1 e y = 1 → ERROR  
  
podría cortarse la ejecución y que ambas variables tengan el valor 1.

Problema cuando las sentencias no cumplen con la prop de a lo sumo una vez,

necesito si o si ejecutarla como si fuese atom aunque incluya muchas sentenciaas nadie deberia interferir, programar un bloque que este seguro q solo ejecutan esa. Sin que nadie pueda ver o laterar las variables comaprtidas con laas que estamos trabajando, las sentencias await,

sinc por exl mutua unicamente, entre < lo que tenemos q ejecutar como grano grueso> no puede haber dos ejecutando esto < >

dentro hacemos ref a la variable compartida a, se va a poder generar una interferencia, la unca manera de asegurar q no se genere la interferencia es que se haga dentro de estas sentencias await.

sentencias atomicas incondicionales, una vez que logra acceder las ejecutará sin inconvenientes

Las sentencias await tb nos permite hacer sentencias atom condicionales, une sinc por e.m con sinc por cond, el proceso con e.m espera hasta que la cond sea verdadera y cuando es verdadera sin salir de la em ejecuta el conj de sentencias.

si es falso, libera la secc criticay leugo entrara hasta encontrar la verdadera. Mientras chequeo q la cond es verdadera nadie en el medio podra modificar el valor dee la condicion

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

La expresión booleanaBespecifica una condición de demora.

S es una secuencia de sentencias que se garantiza que termina.

Se garantiza queB es true cuando comienza la ejecución de S.

Ningún estado interno de S es visible para los otros procesos.

sin condicion saco B, y solo la condicion sin S.   
  
Await general: 〈await (s>0) s=s-1;〉

Await para exclusión mutua: 〈x = x + 1; y = y + 1〉

unicamente sinc por condicion:

• Ejemplo await para sincronización por condición: 〈await (count > 0) 〉

Si B satisface ASV, puede implementarse como busywaiting o spin loop

do (not B) → skip od (while (not B); )  
  
esperar en una est de contro iteratica, hasta q esa cond sea verdadera, deja de ser pseudo codigo, no permite empezar a trabajar con variables compartidas.

incondicionales ---> no utilizan la condicion, el await

cuando usa la cond son condicionales.

**Propiedades de seguridad y vida**

Una propiedad de un programa concurrente es un atributo verdadero en

cualquiera de las historias de ejecución del mismo

Toda propiedad puede ser formulada en términos de dos clases: seguridad y

vida.

Clase 2 30

• seguridad (safety)

Nada malo le ocurre a un proceso: asegura estados consistentes.

Una falla de seguridad indica que algo anda mal.

Ejemplos de propiedades de seguridad: exclusión mutua, ausencia de

interferencia entre procesos, partial correctness.  
  
que pasa con la correccion total, siempre va a terminar y siempre termina bien, union de ambas propiedades.

Fairness, garantizar de q todos puedan avanzar con la ejecucion, una acc atom en un proc es

scheduling determina cual es la proxima q se va a ejecutar para saber cual es la historia q se está armando. Se le da el procesador a un proc y hasta q se libera se le da el uso del procesador.

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.  
  
 iterando, y del otro lado pone en falso, si el ue gana el uso del procesador es while, nunca terminaria.

bool continue = true;   
co while (continue); // continue = false; oc  
  
las politicas de scheduling se clasif en tres:

**Fairness Incondicional**. Una política de scheduling es incondicionalmente

fair si toda acción atómica incondicional que es elegible eventualmente es

ejecutada.

En el ejemplo anterior, RR es incondicionalmente fair en monoprocesador, y

la ejecución paralela lo es en un multiprocesador.  
  
Cualquier acc atom incond, no await o estaba hecho con await pero con booleana, en algun momento se va a ejecutar y como no tiene condicion cuando se le de permiso para ejecutar

Seguro van a poder ejecutarse, qeu pasa con las condicionales? no nos asegura q en algun moemnto pueda ejecutarse, capaz nunca logrre pasar ese punto, no es q no pase pero no estamos seguros.

**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  
  
Nos asegura q se va a poder ejecutar en algun momento, y va a ser ejecutada si la cond se vuelve verdadera y no vuelve a falso hasta poder ejecutarla .  
  
Seguro se va a poder ejecutar en algun momento  
LA que usamos es debil, rr.  **FALTARIA ESTE CASO**

No es suficiente para asegurar que cualquier sentencia await elegible eventualmente se ejecuta: la guarda podría cambiar el valor (de false a true y nuevamente a false) mientras un proceso está demorado.

Mientras la variable cont es v, true y luego f, try true para poner la variable cont en falso  
  
  
**EXP Pract1**  
AWAIT por EXCL mutua. Va a ingresar cualquiera no existe orden cuando la excl mutua se libera

Por condicion, demorarse hasta q la cond booleana sea verdadera.   
  
parc es local asi q no necesito sec crit

total si xq es var compartida

minim el uso de vari compartidas

Como solo lo hace el docente ACTUAL = I podria no ir entre <> porq es uno solo.

necesitamos sincronizar cuando el docente temina de tomar el examen y el alumno puede rendir.

Variable compartida listo q me permite realizar la sincronizacion, docente listo en Trrue para avisarle a alumno(ESTA VARIABLE HAY Q RESERTEARLA para que el alumno se demore y no pase de largo)   
El alumno puede pasarlo a falso, para l proxima iteracion. El docente debe esparar a que listo sea falso,

Ambas sentencia podrian sacarse de la seccion critica.  
  
Ejemplo 4

Cajero ocupado o no-

para ver si hay q encolarse o no-

Si es anciana al principio de la cola.

si no hay nadie ocupa el cajero, por eso me fijo primero si está ocupado o no.

Cuando termina de usar el cajero, y liberarlo(despertar al primero que ya esta)

Cola vacia, liberar poniendo cartel libre.

Solo lo deja desocupado en el caso de que no haya nadie esperando.

Los UNICOS procesos son las personas, cajero RECURSO compartido.   
  
//si el cajero está libre lo usa, sino se encola

//usa el cajero

//libera el cajero  
  
Si llegan y mas de uno se quiere agregar.. que pasa??

podría pasar una interferencia, o se podrian encolar al revés, necesitamos que el trabajo por esa cola sea con exclusion mutua, NO PUede haber otro proceso ejecutando el agregar o el dsaar.

¿Cómo realizamos la espera?

Variable que nos indique quien es la sig persona que tiene qe pasar al cajero,

siguiente=-1

await siguiente=id

entonces el proceso se demora

cuadno siguiente es ese id, es su turno y puede utilizar el cajero automatico.

Como libero el cajero y despierto al primero de la cola.

el valor de id tome el valor del proceso q esta primero en la cola

saco el primer elem de la cola y siguiente va a tomar ese valor.

al estár trabajando sobre la misma cola, manejo dentro de una secc crit.

¿q ocurre?

la cola c está vacia? sacar de una cola vacia?

a quien despierto? no hago un sacar, lo tengo que marcar como que el cajero está libre, entocnes le pongo -1 de vuelta.

antes de ponerme en la cola tengo q fijarme si el cajero está ocupado,

si siguiente =-1 no hay nadie, en sig pongo el valor de mi id y listo. siguiente:= id.

si tiene tiene otro valor está ocupado.

unicamente en la cola si estaba ocupado, para exc mutua, en forma atomica ejecuto todo el if, pero de acuerdo al valor de sig en ese instante hago una cosa u otra, pero no es que me demoro. me demoro en el await .

Rec compartido donde debe usarse una persona a la vez y los procesos admin el acceso al rec compartido.

**DEFINICIONES IMPORTANTES**

No hay un orden preestablecido en la ejecución ⇒ no determinismo

(ejecuciones con la misma “entrada” puede generar diferentes “salidas”)

No **determinismo**, en ejecución concurrentes con la misma entrada se pueden obtener resultados diferentes *depende del orden en que se van ejecutando las cosas*.

Algunas **ventajas de la programacion concurrente**

- La velocidad de ejecución que se puede alcanzar.

- Mejor utilización de la CPU de cada procesador, de los cores.

- Explotación de la concurrencia inherente a la mayoría de los problemas reales, soluciones más sencilla de pensar/ver.

Un Programa concurrente va a estar formado por varios procesos, este programa es para resolver un único problema, ese programa va a estar compuesto por varios procesos, distintos comportamientos, pueden ser independientes los procesos cada uno resuelve su parte sin interactuar, pero vemos cuando estos procesos cooperan o compiten, entonces los procesos puede ser de tres formas, **independientes, cooperar o competir**

**Cooperar** c/u ejecutar una parte y después de alguna manera esas partes se integran para obtener el resultado final.

**Compiten** pelean para utilizar cierto recurso compartido(venta de pasaje de avion, usuarios compiten para acc a cierto pasaje)

Esto no quiere decir que un proceso todo el tiempo compite, ni todo el tiempo coopera ni es independiente, esto puede ir variando.

Ejemplos abuelitas.

**Independiente** lo que menos nos interesa en la materia, tres abuelas, cada una de las abuelas va a tejer su propio pullover y agujas.

**Competencias**: cierto recurso compartido, comunes a los procesos o subconjunto de procesos que tienen que trabajar con ello, si ese recurso podría ser usado por más de un proceso a la vez no habría problemas, cuando puede utilizarlo uno solo compite por el recurso.

Ahora dos abuelas, compiten por el ovillo de lana uno blanco y uno gris, cuando quieren usar el mismo, ver en qué caso utiliza cual, compiten por el uso de los ovillos, vamos a tener que secuencializar, termine la otra y se lo dé a la abuela.

Problemas:

**DeadLock** → ninguno suelta su recurso hasta no obtener el otro y se produce deadlock(punto muerto), abrazo mortal porque ninguno proceso va a poder continuar.

**Inanición →** alguno de los procesos nunca logre conseguir el recurso porque el otro gana, cuando más procesos hay es más probable que esto se dé.

**Cooperación →** dividimos el problema en partes y luego lo unimos para formar lo que sea el resultado. Hay muchas decisiones que tomar, usamos 4 , o 5 tres que procesen y una que una? el que terminó antes? el más rápido? Lo mejor posible en cuanto al tiempo de ejecución.

Los procesos se comportan de las tres maneras, pueden intercalarse entre ellas.

Otro enfoque: un sóla máquina dedica una parte del tiempo a cada

componente del objeto ⇒ Concurrencia sin paralelismo de hardware ⇒

Menor speedup.

Dificultades ⇒

Distribución de carga de trabajo.

Necesidad de compartir recursos.

Necesidad de esperarse en puntos clave.

Necesidad de comunicarse.

Necesidad de recuperar el “estado” de cada proceso alretomarlo.

**CONCURRENCIA Concepto de software no restringido a una arquitectura particular de hardware ni a un número determinado de procesadores.**

**HILOS Y PROCESOS**  
Al tener el ED de un mismo proceso compartido tiene sus inconvenientes, porque podrá haber dos hilos modificar el valor de una variable al mismo tiempo → **interferencia**.

Necesito algún mecanismo que me permita evitar las interferencias, sino resultado erróneos, distinto a no determinismo→ dos result dif. La concu puede estar provista por el lenguaje como en Java, o provista por el SO, ciertas librerías y a través de eso el SO se encarga de generar la concurrencia.   
 **COMUNICACIÓN**  
**En algún momento puede necesitar cierto dato que tiene otro de los procesos.** Comunicación entre procesos, modo en que se organizan y transmiten los datos, requiere especificar ciertos protocolos, como progresar y que se haga de forma correcta.

En la clasificación en dos grandes grupos,

Los procesos se COMUNICAN: • Por Memoria Compartida.

• Por Pasaje de Mensajes.

• **Por Memoria Compartida:** nos permite diferenciar la forma en que se programa y se trabaja, necesito si o si algún lugar de memoria donde todos los proc pueda acceder, dejando los datos en alguna dirección donde otro pueda tomarlo, todo a través de esa

memoria compartida, una físicamente y además los procesos acceder al mismo ED. Cuidado con el uso dos procesos quieren modificar la misma variable por ejemplo. Necesito que la herramienta me brinde algún bloqueo o liberar el acceso a una posición de memoria. Lo que se usa semáforo, nos habilitará o no el paso. Una máquina que tenga una memoria compartida, que permita que los procesos puedan acceder al mismo espacio de direcciones.

• **Por Pasaje de Mensajes.** En las unidades de procesadores separadas, no existen memoria compartida, la comunicación enviando mensajes a través de una red, arquitectura distribuida, procesos separados, también utilizado en memoria compartida y esos canales se simulan por espacio de memoria pero los procesos no pueden acceder al mismo espacio de direcciones, cada uno tiene su ED, el único que conoce eso es el dueño de los datos y para compartir esa info lo hace a través de mensaje. Canales lógicos o físicos que me permiten transmitir, tengo que poder definir esos canales, sentencias primitivas para los datos que se envían a través de esos canales, a quien le envió, de quien recibo información.

**SINCRONIZACIÓN**

La sincronización es la posesión de información acerca de otro proceso para

coordinar actividades. Los procesos se sincronizan:

• **Por exclusión mutua**. Asegurar que sólo un proceso tenga acceso a un recurso compartido en un instante de tiempo solo por un proceso a la vez, y hasta que no termine no cambia. Ejemplo: Una variable compartida que tiene que ser modificada, el uso de una impresora. Si el programa tiene secciones críticas que pueden compartir más de un

proceso, la exclusión mutua evita que dos o más procesos puedan encontrarse

en la misma sección crítica al mismo tiempo.

**• Por condición.** Permite bloquear la ejecución de un proceso hasta que se cumpla una condición dada. Ejemplo: elemento que hay en una cola, no puedo ir y sacarlo espero que este vacía para recién hacer el pop y sacar el elemento.

*Ejemplo de los dos mecanismos de sincronización en un problema de utilización de un área de memoria compartida (buffer limitado con productores y consumidores).*

Problema concurrente que debe utilizar ambos, la utilización de memoria compartida, al cual todos los proc pueden acceder y es limitada, procesos productores y consumidores, no puedo dejar elementos hasta que alguien lo haya dejado, los consumidores tampoco pueden tomarlo hasta que al menos haya un dato en el buffer, se dan ambos tipos de sincronización, el consumidor se bloquea hasta que hay algo y es consumido, se bloquea hasta que haya espacio para dejar un elemento, esto es por condición.

Si hay más de un productor, los productores van a tener que sincronizarse entre ellos por exclusión mutua para que de los productores uno solo pueda acceder.

Del otro lado tenemos la sincronización por exclusión mutua, si hay más de un productor para dejar un elemento pero un único elemento, por exclusión mutua van a tener que sincronizarse.  
 **INTERFERENCIA**

Un proceso toma una acción que invalida las suposiciones de otro hechas por otro proceso.

**PRIORIDAD Y GRANULARIDAD**

Un proceso que tiene mayor **prioridad** puede causar la suspensión(preemption) de otro proceso concurrente.

Análogamente puede tomar un recurso compartido, obligando a retirarse a otro proceso que lo tenga en un instante dado.

La **granularidad** de una aplicación está dada por la relación entre el cómputo y la

comunicación.

Relación y adaptación a la arquitectura.

Grano fino cantidad de cómputo, y comunicación, poco cómputo y comunicar y grano grueso código grande y luego comunicado, bloque de trabajo con poca comunicación.

**MANEJO DE LOS RECURSOS**

Uno de los temas principales de la programación concurrente es la administración de recursos compartidos:

• Esto incluye la **asignación** de recursos compartidos, métodos de acceso a los recursos, bloqueo y liberación de recursos, seguridad y consistencia.

• Una propiedad deseable en sistemas concurrentes es el equilibrio en el acceso a recursos compartidos por todos los procesos (**fairness**).

• Dos situaciones NO deseadas en los programas concurrentes son la **inanición** de un proceso (no logra acceder a los recursos compartidos) y el **overloading** de un proceso (la carga asignada excede su capacidad de procesamiento).

• Otro problema importante que se debe evitar es el **deadlock**.  
Dos (o más) procesos pueden entrar en deadlock, si por error de

programación ambos se quedan esperando que el otro libere un

recurso compartido. La ausencia de deadlock es una propiedad

necesaria en los procesos concurrentes.

**• 4 propiedades necesarias y suficientes para que exista deadlock son:**

* Recursos reusables serialmente: los procesos comparten recursos que pueden usar con exclusión mutua.
* Adquisición incremental: los procesos mantienen los recursos que poseen mientras esperan adquirir recursos adicionales.
* No-preemption: una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente.
* Espera cíclica: existe una cadena circular (ciclo) de procesos tal que cada uno tiene un recurso que su sucesor en el ciclo está esperando adquirir.

**• Requerimientos de un lenguaje de programación concurrente:**

* Indicar las tareas o procesos que pueden ejecutarse concurrentemente.
* Mecanismos de sincronización.
* Mecanismos de comunicación entre los procesos.

**PROBLEMAS DE LOS PROGRAMAS CONCURRENTES**

* Los procesos no son independientes y comparten recursos. La necesidad de

utilizar mecanismos de exclusión mutua y sincronización agrega complejidad

a los programas.

* Los procesos iniciados dentro de un programa concurrente pueden NO estar

“vivos”. Esta pérdida de la propiedad de liveness puede indicar deadlocks o

una mala distribución de recursos.

* Hay un **no determinismo** implícito en el interleaving de procesos concurrentes.

Esto significa que dos ejecuciones del mismo programa no necesariamente

son idénticas ⇒ dificultad para la interpretación y debug.

* Posible reducción de performance por overhead de context switch,

comunicación,sincronización.

* Mayor tiempo de desarrollo y puesta a punto. Difícil paralelizar algoritmos

secuenciales.

* Necesidad de adaptar el software concurrente al hardware paralelo para

mejora real en el rendimiento.  
  
La performance del context switch, cuando saco un proceso tengo que de alguna manera guardar el estado para cuando vuelvo guardar su estado todo este context switch lleva su tiempo.  
  
 **CONCURRENCIA**  
Concepto de software no restringido a una arquitectura particular de hardware ni a un

número determinado de procesadores.

Especificar la concurrencia implica especificar los procesos concurrentes, su comunicación y su sincronización.  
 **PARALELISMO**

Se asocia con la ejecución concurrente en múltiples procesadores con el objetivo principal

de reducir el tiempo de ejecución.  
  
**Clases de instrucciones**

**Asignación**

Asignación simple x = e

Sentencia de asignación compuesta   
 x:=x+1;  
 y:=y-1;  
 a[2]=6 ,   
 aa[2,6]= a[4]  
  
Llamado a funciones x=f(y)+g(6) - 7

swap: v1:=:v2 → intercambiar los valores de dos variables

skip: termina inmediatamente y no tiene efecto sobre ninguna variable de programa → si utilizamos, no hace nada no utiliza procesador no hace cambios.   
  
Sentencia **co**:

**co** S1 // ..... // Sn oc → Ejecuta las Si tareas concurrentemente.

La ejecución del co termina cuando todas las tareas terminaron.

Cuantificadores.

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

**Process:** otra forma de representar concurrencia

processA {sentencias} → proceso único independiente.

Cuantificadores.

process B [i=1 to n] {sentencias} → n procesos independientes.

Diferencia: **process** ejecuta en background, mientras el código que contiene un **co** espera a que el proceso creado por la sentencia co termine antes de ejecutar la siguiente sentencia.  
  
  
**ATOMICIDAD DE GRANO FINO**  
**Estado** de un programa concurrente, el valor que tienen las variables compartidas de cada proceso en cierto momento, si uno corta la ejecución de los proceso y guarda esos valores ese sería el estado de un programa.

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

Una acción atómica hace una transformación de estado **indivisibles** (estados intermedios invisibles para otros procesos).

• Ejecución de un programa concurrente → intercalado (interleaving) de las acciones atómicas ejecutadas por procesos individuales.

• Historia de un programa concurrente (trace): ejecución de un programa concurrente con un interleaving particular. En general el número de posibles historias de un programa concurrente es enorme; pero no todas son válidas.

• Interacción → determina cuáles historias son correctas.  
  
**La sincronización por condición permite restringir las historias de un programa concurrente para asegurar el orden temporal necesario.**  
Atomicidad de grano fino   
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.

**Referencia crítica en una expresión ⇒ referencia a una variable que es modificada por otro proceso.**

**Asumamos que toda referencia crítica es a una variable simple leída y escrita**

**atómicamente.  
  
Puede haber a lo sumo una variable compartida, y puede ser referenciada a lo sumo una vez**