**Concurrencia**

**¿Qué es?**

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

**¿Dónde podemos ver que hay concurrencia?**

Desde los navegadores web, accediendo a distintas páginas al mismo tiempo, respondiendo y atendiendo al cliente que escribió algo, en el acceso a disco, los teléfonos(mientras estoy en una llamada recibo mensajes), sistema de venta de pasajes, muchas personas al mismo tiempo para reservar comprar su pasaje accediendo a un mismo conjunto de información.

Interactúan entre sí para decidir qué pasa, y entre todos resuelven algo, o cuando hay algún problema actúan, y concurrencia pero no programa concurrente o si es un programa paralelo dentro de una concurrencia.

Algo en la realidad que sea secuencial es dificil de encontrar, que no esté integrado por componentes que se estén actuando al mismo tiempo, el problema cuando hablamos o tenemos que replicar algo del mundo real, en vez de pensarlo concurrentemente como en la realidad es, lo transformamos a un algoritmo secuencial que es el que vamos a resolver, esto va a traer inconvenientes, es más complejo tratar el problema concurrente resolverlo de forma secuencial y no estaríamos aprovechando la potencia de las máquinas de forma concurrente.

**Ejemplo**

Si lo pensamos de forma concurrente, es más sencillo. Cada tres segundos imprimir en pantalla un cartel rojo, solución secuencial loop infinito demoro 3 segundos muestro cartel.

¿Qué pasa si además ahora hay un cartel azul cada 5 seg? A los tres uno rojo, dos segundos azul, al segundo uno rojo.

¿Qué pasa si sumo otro? Tratar de resolverlo directamente de forma concurrente, ya que todos los carteles son independientes. Un pedacito de código en el cual vamos a tener un color de cartel y el tiempo que hay que esperar, con distintos parámetros.

| Programa Cartel (color, tiempo)  Mientras (true)  Demorar (tiempo segundos)  Desplegar cartel (color)  Fin mientras  Fin programa |
| --- |

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.

**¿Por qué la importancia y la necesidad de la programación concurrente?**

Si le pongo mas core si la ejecución del programa es secuencial, si cada core no voy a reducir, si yo quiero reducir el tiempo de ejecución no me queda otra que la aplicación sea concurrente y pueda aprovechar el multicore, es la única forma en que yo podría bajar el tiempo de ejecución de ese programa.

Hay veces que sería muy complejo si no tuviera la concurrencia hacer un programa secuencial, avión lleno de sensores que pasa si los miro de forma secuencial, 1 que te dio, 2 que te dio, sensor 3, el 1 tiene un problema tengo que esperar a que terminen todos para ver el error, no puedo el avión se cayó, porque si tengo 10000 sensores y el tres tuvo un error lo tengo que resolver en el momento no cuando los otros ya terminaron.

También es importante la concurrencia para *aprovechar los tiempos muertos*, E/S, la memoria para aprovechar y continuar trabajando en esa aplicación. Todo esto nos mejora el tiempo final de la ejecución.

**Sistemas distribuidos**, accesos a un cierto sistema, comparten algún tipo de recursos, una BD, y desde distintos lugares, pueden acceder a través del mismo sistema distribuido, para acceder a esa base de datos por ejemplo para comprar un pasaje de avión.

**Objetivos de los sistemas concurrentes**

Principales *ajustar el modelo de arquitectura de hardware y software* que desarrolle al problema del mundo real a resolver, arquitectura donde ejecutó más el programa performante reducir el tiempo de ejecución de la aplicación que esté desarrollando aprovechando las ventajas desde el punto de vista de HW y SW, no siempre los objetivos son los mismos, en el pasaje no es tan importante mejorar el rendimiento porque eso está ajustado más que nada a resolver el problema del mundo real.

Algunas **ventajas**

- 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.**

Procesos los llamaremos, secuencial único flujo de control, una instrucción tras otra, único hilo de ejecución, un proceso de uno de esos programas secuenciales, un hilo de ejecución, llamaremos proceso a cualquiera de los componentes que forman 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 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** → 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.

**Diferencia entre Procesamiento secuencial, concurrente y paralelo.**

Analicemos la solución **secuencial** y monoprocesador (una máquina) para fabricar un objeto compuesto por N partes o módulos.

Esa máquina tiene que generar cada uno de los N módulos y luego se crea con esos módulos que esa máquina fabrica.

Secuencial, una única máquina, se hace uno, termina, el segundo, término el tercero. La soluc sec nos fuerza a determinado orden secuencial estricto. Esperar a que N piezas estén terminadas para poder hacer el ensamblado y terminar.,1 hora cada módulo, n horas, mas el ensamblado(2 hs) → n + 2 hs

**Paralelismo**:Si tuviésemos N máquinas, no hay necesidad de competir por la materia prima, de esta manera cada máquina al mismo tiempo va a entrar fabricando el módulo que le tocó, en un unidad de tiempo determinado, todos van a estar terminados y se pueden ensamblar las distintas partes, mucho más rápido comparado con lo anterior, solución paralela que trabajan varias a mismo tiempo. Cada módulo media hora, N horas, más el ensamblado(2 hs), 1 hora de trabajo las n partes + 2 hs, 3 hs total. Solucion mucho mas rápida la paralela que la secuencia,pero costoso tener N máquinas donde va a trabajar poco tiempo cad una, consecuencia, menor tiempo para completar el trabajo, menos esfuerzo individual, cada máquina trabaja menos tiempo, menos desgaste por ende dura más tiempo para armar más número de objetos, paralelismo de HW, máquinas que nos permite estar resolviendo al mismo tiempo más de una. Cosas a tener en cuenta para que esto se pueda , dist de carg de trabajo, no todas iguales, módulos más chicos o mas grandes, de estas partes mas tiempo menos tiempo lo más importante de un proceso paralelo tratar de distribuir el trabajo de la forma que se adapte mejor a las máquinas que voy a trabajar. Principal cosa para cumplir con los requisitos, buena distribución del trabajo de las máquinas que forman parte de la solución.

Compartir ciertos recursos evitando conflictos, recurso compartido por todas las máquinas, si dos tienen un problema, el recurso es uno, ¿como resuelvo? una necesita más tiempo de trabajo, elijo esta, por prioridad, si un es más costosa dinero, primero resuelvo esa.

-Necesidad de compartir recursos evitando conflictos.

*-Necesidad de esperarse en puntos clave*, esperar un proceso a que termine y me mande ciertos info.

-*Necesidad de comunicarse,* relacionado con el punto anterior.

-*Tratamiento de las fallas*, si se rompe podría no poder resolverlo en el momento, podría manejar un tratamiento de fallas trasladarlo a otra máquina, podría tardar más pero no detendrá la “producción”, ver cómo se redistribuye el resto de las maquinas.

-Asignación de una de las máquinas para el ensamblado (¿Cual?). quien va a hacer el trabajo, la mas rapida, la que antes esta desocupada o la uqe hizo la pieza mas chica y trabajo menos? o la que esta en el centro y la distancia es la misma a todas. La que mas convenga para el rendimiento y esfuerzo.

Si tuviera una sola máquina, aprovechar los tiempos muertos(que no se trabaja sobre el modulo), para ir avanzando en otro de los módulos, así vamos alternando el uso de la máquina entre las distintas piezas. Esto se llama **concurrencia sin paralelismo de Hardware.** Porque el procesamiento es concurrente pero la máquina es una sola. Se logra reducir el tiempo, al aprovechar los tiempos en que la máquina estaba ociosa. Ambos casos son concurrentes, con paralelismo o no.

Dificultades ⇒

* Distribución de carga de trabajo, menores que con paralelismo porque la maquina es una sola y siempre las mismas características, defino solo el orden y cuando pongo uno y otro para hacer el context switch.
* Necesidad de compartir recursos, menor el problema, la máquina es una sola.
* Necesidad de esperar en puntos clave.
* Necesidad de comunicarse, va a cambiar el mecanismo, si comparten un lugar puede cambiar la forma porque están en el mismo lugar.
* **Necesidad de recuperar el “estado” de cada proceso al retomarlo,**  se agrega acá, termine la etapa uno de un módulo, para la tapa dos ubicarlo en donde corresponde, volver a armar par que el módulo que le toca trabajar este preparado y pueda resolverlo eficientemente, recuperar el estado como estaba antes, esto *consume tiempo,* lo ocupa este context switch.

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

**No está restringido a donde lo voy a ejecutar, si es única unidad de proceso o multi.concepto de sw el hecho de div el programa que tengo en procesos que van a competir o cooperar entre ellos para resolver el problema**

El So concurrencia sin paralelismo de HW, varios procesos que corren en una unidad de procesamiento , el so le da un poco de tiempo a cada proceso para que ejecute sobre la unidad de procesamiento en si pasado ese tiempo, o cuando termina, se lo libera o lo saca y se deja ejecutar a otro proceso listo, context switch entre los procesos. **Copiar el estado final del proceso que estaba ejecutando en el reg del procesado en una memoria local para el donde va a estar guardado mientras el proceso está suspendido hasta que le vuelvan a dar el procesador. Cuando el proceso que va a empezar a ejecutar en la unidad de proceso toma los datos guardados, los pone en el registro del procesador y empieza a ejecutar desde el punto que haya terminado el anterior, asi funciona context switch.**

**Si se bloquea porque espera se lo manda a otra cola de proceso que no está listo para ejecutar , cuando está listo pasa a la cola de listos y ahí pasa para poder luego ejecutar. Esto lleva tiempo, si es posible no conviene tener muchos.**

1. un cachito cada uno

2. cc una unidad de procesamiento para cada proceso

3. una cierta cantidad de procesos N y una cantidad menor de unidades de procesamiento, más de una pero 2 a n-1 en uno o más vamos a tener más de un proceso ejecutando.

Además de definir las porciones de código que se van a ejecutar en forma simultánea, proc o tareas, características importantes, la interacción entre los procesos, los puntos de espera, estas interacciones consumen un cierto tiempo.

NO determinismo, mismo prog con las mismas entradas con resultado distinto, por ahí variaron la forma en que se ejecutaron las instrucciones, con resultado válidos en ambos casos distintos pero válida ambas soluciones, el no determinismo nos ocasiona dificultad para entender la ejecución del programa, debug(una vez cada tanto resultado erróneo).

**HILOS y PROCESOS**

Los **procesos** son entidades mas grandes dentro del so, c u tiene su propio espacio de direcciones(ningún otro accede y ve su información) y recursos, su propia paginación.

Los **hilos**, thread procesos que tienen contador de programa y pila de ejecución pero no tiene contexto pesado, estos hilos pertenecen a un proceso, el proceso tiene lo pesado tabla de paginación, y el hilo info más local de lo que nec para ejec, Espacio de Direc del proceso al cual pertenece a otro proceso al cual pertenece, si un proceso tiene 25 hilos los 25 hilos acceden a ese Espacio Direcciones. Son más livianos, los context switch es mas fácil hacerlo entre dos hilos que pertenecen al mismo proceso. 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.

Audio 2

**Conceptos Básicos de concurrencia**

**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 envio, de quien recibo información, importante, a pesar de que está pensado a compus separadas conectadas a través de una red de interconexiones los procesos tb podrían estar en una monoprocesador o multicore, canal lógico en vez de red.

**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. El uso de un recurso compartido, usado por un unico proceso a la vez. 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 vacia 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 utilizac de mem compartida, al cual todos los proc pueden acc y es limitada procesos productores y consumidores, no puedo dejar elementos hasta que alguien lo haya dejado, los consumidores tmp puede 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 el consumidor se bloquea hasta que haya espacio para dejar un elemento, esto es por condición.

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

Del otro lado tenemos la sinc 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.

**Problemas que pueden ocurrir en un programa concurrente.**

**Interferencia:** un proceso toma una acción que invalida las suposiciones hechas por otro proceso. Un proceso toma el estado y resulta que hay otros procesos que modificaron ese estado.

Ejemplo 1: variables compartidas, evitar la div por cero, chequea que sea distinto de 0, podría ocurrir que el proc 2 chequea y se da cuenta que es distinto de cero, pero antes modifica hacer la división el proceso a1 lo convierte en 0 y recien despues hace la división, generando un error, A1 invalida a A2 por interferencia que acá produce un error del programa.

Ejemplo 2 Si no utilizo excl mutua para trabajar con la variable, podría pasar q en algun momento ambos a la e están intentando incrementar, los dos quieren incrementar y se haría solo una vez el incremento, deberá haber incrementado dos, uno por cada uno, la interferencia acá yo trabajo con el valor de público sin embargo otro proceso lo modifico y me dej a mi trabajando con otro proceso.

**Prioridad**

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.

El proceso de menor prioridad, es sacado, se suspende su ejecución, puede ser con el procesador o con cualquier otro recurso compartido. obligarlo a salir para ser utilizado pro el de mayor prioridad.(en la pract no va a quitar el proc)

**Granularidad de una aplicación**

Granularidad fina cantidad de cómputo, y comunicación, poco cómputo y comunicar,. grano grueso, codigo grande y luego comunicado, bloque de trabajo con poca comunicación. Tiene que ver la máquina sobre la que esté trabajando.

tengo un pc, potente el procesador y lenta la comunicación, no me conviene una solución de grano fino, en un caso de esa arquitectura que es de grano grueso, me conviene que cada trozo haga muchas operación y se comunique menos.

arquitectura distribuida, grano grueso.

**Admin de rec compartidos**

• Esto incluye la asignación de recursos compartidos a procesos, métodos de acceso a los recursos, bloqueo y liberación de recursos, seguridad y consistencia. ej, cajero automático entre los procesos usuarios deciden, en qué momento y de qué manera

• Una propiedad deseable en sistemas concurrentes es el equilibrio en el acceso a recursos compartidos por todos los procesos (**fairness**). Si todos tiene que usar cierto recurso, que las posibilidades de acceder sean equitativas.

• Dos situaciones NO deseadas en los programas concurrentes son la **inanición** de un proceso (no logra acceder a los recursos compartidos). Por ejemplo acceder por orden alfabético, y el **overloading** de un proceso (la carga asignada excede su capacidad de procesamiento). muchas mas tareas a un proceso de las que puede resolver, o mas trabajo que al resto de los procesos, sobrecargo uno.

• Otro problema importante que se debe evitar es el **deadlock.** Dos o mas procesos no pueden continuar su ejecución porque estan esperando cierto recurso compartido al que no pueden acceder porque lo tiene otro.

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. La propiedad debe cumplirse durante toda la ejecución y en todo momento, si se puede llegar a dar ese deadlock nadie puede seguir.

**4 propiedades necesarias y suficientes para que exista deadlock son: La ausencia de deadlock es necesaria en los programas concurrentes**

**Recursos reusables serialmente:** los procesos comparten recursos que pueden usar con exclusión mutua.Lo utiliza a medida que se va liberando pero con exclusión mutua

**Adquisición incremental:** los procesos mantienen los recursos que poseen mientras esperan adquirir recursos adicionales. Recurso libre lo toma, otro libre lo toma, va tomando de a uno lo que necesita.

**No-preemption:** una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente.Nadie lo puede forzar a dejarlo, una vez que terminó, no forzarlo.

**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. Necesita un recurso compartido que tiene en siguiente proceso, a el de b, b el d c, c el de a.

*Cuatro prop que se deben dar para que exista deadlock, yo debo asegurarme que alguna no se dé, con que una de esas cuatro no se dé no podría existir.*

**Requerimientos de un lenguaje de programación concurrente**

Indicar las tareas o procesos que pueden ejecutarse concurrentemente(a través de procesos, hilos pero definir cuáles módulos formarán parte de esto).

Mecanismos de sincronización.(para exclusión mutua o condición)

Mecanismos de comunicación entre los procesos(para ambos)

Debe brindarme todas las posibilidades de sincronización y comunicación.

ADA o java. O librerías cobra openMP.

**Problemas asociados con la prog concurrente**

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 prog concu no existan ya, y que quieran interactuar con ellos y los otros procesos no sepan, interactuar con ellos cuando no están, esta pérdida de la vida,porque se bloquearon pro deadlock, o terminó bien y se quedó sin trabajo. Esto indica una mala distribución de los procesos.

No **determinismo** no todas las instrucciones del mismo proceso resultado finales o intermedios diferentes que hacen difícil la interpretación de estos programas o el debug, si quiero ver el error

LA performance, 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

overhead lo que es la sincronización y la comunicación

sincro por condición también, esperar a que lo haga, comunicación si es por pasaje de mensaje tiene cierto coste por más de que no tenga que esperar y tb la sicn por exclusión mutua, el hecho que se scuencialice todo lo que tiene q hacerse por exclusión mutua, cierto retardo.

Mayor tiempo de desarrollo y puesta a punto(tratar de lograr el mayor rendimiento posible y aprovechar lo mas posible el HW para reducir el tiempo de ejecución de la aplicación).

Difícil paralelizar algoritmos secuenciales, a veces imposible, aplic secuenciales muy optimizadas imposible paralelizarlas, hay que buscar otra solución secuencial al mism problema.

Necesidad de adaptar el software concurrente al hardware paralelo para mejorar el rendimiento.

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.

Especificar los procesos que van a formar parte del programa, cómo se comunican y cómo se van a sincronizar. Ese SW en un mono o en una arquitectura con más de una unidad de procesamiento. Paralela o monoprocesador. el prog sw a concurrencia en gral .

PARALELISMO ⇒ Se asocia con la ejecución concurrente en múltiples procesadores con el objetivo principal de reducir el tiempo de ejecución.

Una parte dentro de lo que es la concurrencia, agarra uno de esos programas y lo ejecuta sobre una arquitectura que tiene más de una unidad de procesamiento, ya es paralela la ejecución, en general objetivo del paralelismo **reducir el tiempo final de ejecución** tratando de mantener un buen rendimiento usando la arquitectura completa.

**Concurrencia a nivel de HW**

Límite físico en la velocidad de los procesadores

• Las máquinas monoprocesador ya no pueden mejorar. Surgen los multicore porque no pude mejorar más cada unidad, agrego mas. 2,4,8 y así. La única forma de poder aprovecharlo era que los prog sean concurrente

• Más procesadores por chip para mayor potencia de cómputo.

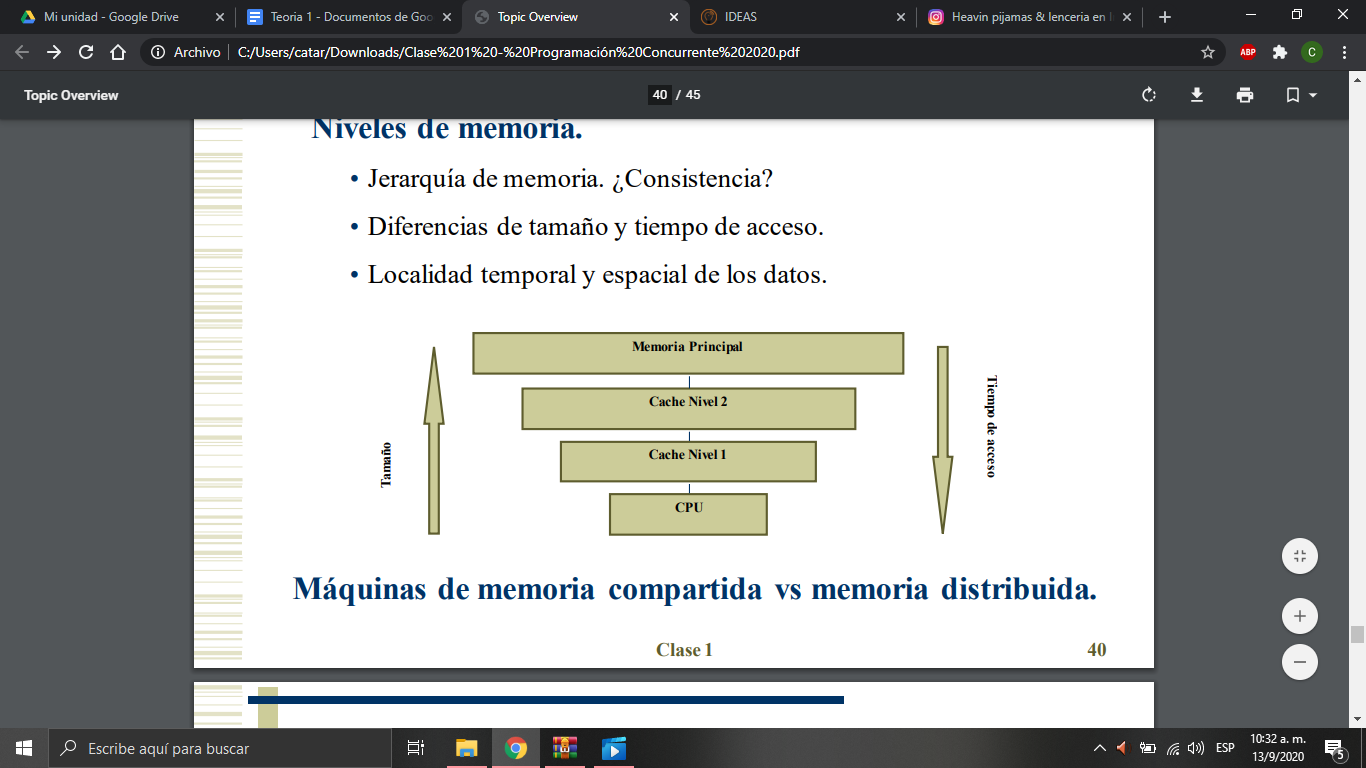
• Multicores → Cluster de multicores → Consumo. Actividades paralelas unimos, más hw y aplic, y nuevos paradigmas, máquinas de memoria compartida y distribuida, para procesos en multicores separados pero unidos por la red de interconexión, híbridos, mezclar comunicación de pasaje de mensaje con comunicación de memoria compartida. Nueva área.

• Uso eficiente → Programación concurrente y paralela

Nuevas áreas de investigación y de trabajo, y nuevos problemas, el consumo energético es uno, y hoy en día adquiere más importancia el consumo, incluido como parte de las métricas.

Aprovechar esto la programación concurrente y paralela, en este caso siempre paralela, las máquinas ya tienen más de una unidad de procesamiento. Para que aprovechan de la forma más adecuada posible la arqui sobre la cual está ejecutándose.

procesador, distintos niveles de cache, mem principal, cuando más cerca del reg de la cpu el tamaño de la memoria y la velocidad de acceso tb es menor.



Cuando hablamos de prog concurrente los problemas se incrementan. Monoprocesador, tengo un nivel de caché todo esa caché va a estar disponible para el proceso que estoy corriendo, si logro acceder a la caché y no voy a memoria a buscar los datos reduzco el tiempo de ejecución. si tuviera 4 procesos de una api concurrente y trabajo con muchos datos, los datos de caché son algunos que han sido accedidos y utilizado por estos procesos, el uso de esa caché no va a ser tan bueno.

**Localidad temporal** , si en un momento traigo un dato debería tratar de trabajar lo más posible con ese dato. Todo lo que tenga que hacer con ese dato lo hago ahora.

**Localidad espacial**, cuando traigo un dato traigo un pedacito, lo traje porque trabajo con el primero pero ya que me traje aprovechar los cuatro y resuelva lo que tenga que hacer con los cuatro.

Competencia entre los procesos que están en los distintos cores que acceden a una caché.

Modifico el valor en otra caché de nivel 1, que pasa, el de la mem principal por ejemplo, estas máquinas de mem compartida, protocolo de coherencia de cache para tener consistencia. En estos sistemas vamos a tener un sistema de coherencia que nos permita trabajar con datos correctos cuando uno cambia el de otro, en monoprocesador no tenemos este problema.   
  
Clasificar las máquinas paralelas en dos opciones contrarias pero en el medio la combinación de ambos

Arquitectura: de mem compartida, entre las unidades de procesamiento que la formen.

y las de memoria distribuida, tenemos distintas unidades de procesamiento, cada una tiene su propia memoria y no puede acceder a la de otro.

multicore para la materia.

distribuida → cluster de monoprocesadores, red de interconexión.

ahora tenemos cluster de multicores, máquina de mem distribuida y compartida.

**Multiprocesadores de memoria compartida.**

Las unidades de procesamiento van a poder acceder, y dejar o sacar cosas de ahi, comunicarse a través de esa memoria compartida. Cada una podria tener su propia unidad de procesamiento y una red que las comunica.

• Interacción modificando datos en la memoria compartida.

• Esquemas UMA con bus o crossbar switch (SMP, multiprocesadores

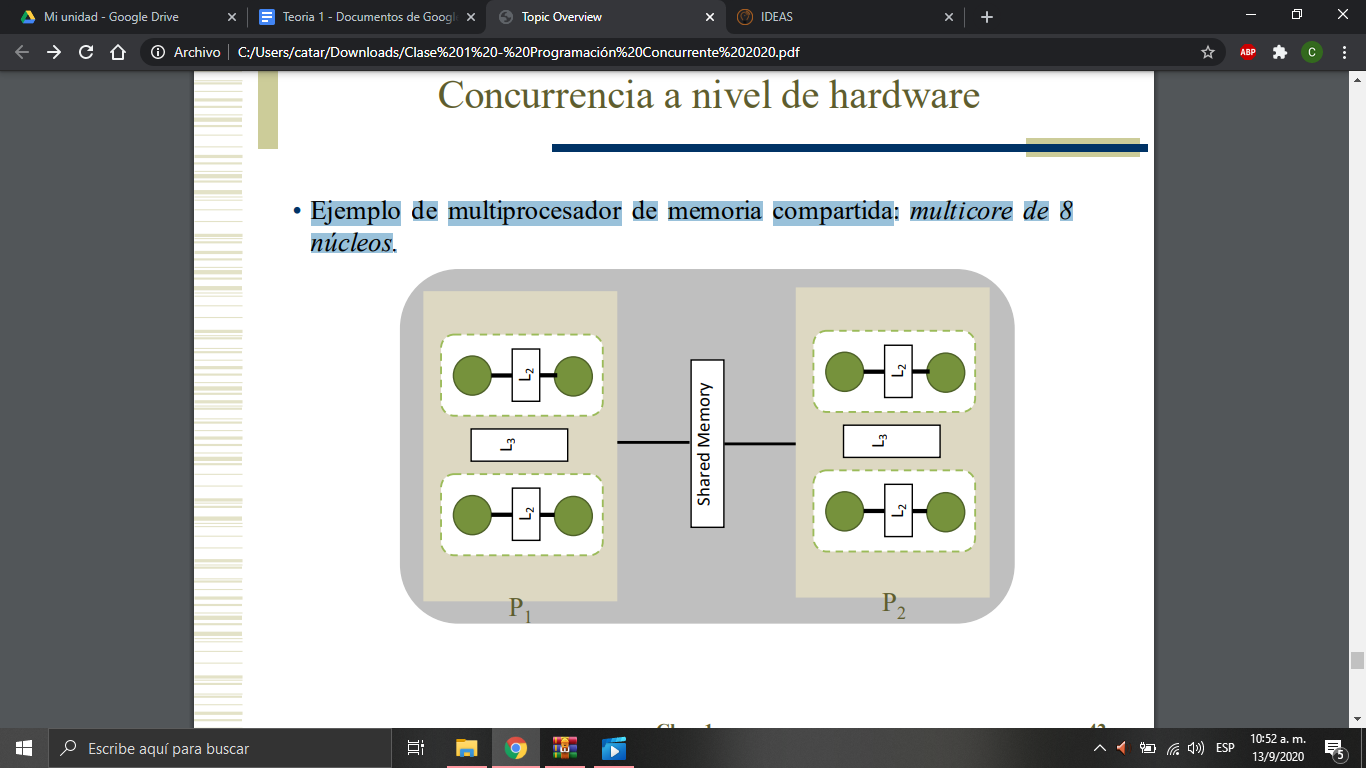
simétricos). Problemas de sincronización y consistencia. Todas las unidades de procesamiento tardan los mismo en acceder a la memoria compartida.

• Esquemas NUMA para mayor número de procesadores distribuidos. Acceso a memoria no uniforme, me conviene que los datos a los que tengo que acceder este en un bloque de memoria que esté ahí.

• Problema de consistencia.

Ejemplo de multiprocesador de memoria compartida: multicore de 8 núcleos.

dos procesadores con dos bloques, y cad uno dos cores, comparten una cache nivel L3, entre los dos



**Multiprocesadores con memoria distribuida.**

• Procesadores(máquina o servidor) conectados por una red de interconexión, puede ser internet, o máquinas acopladas, en un gabinete conectadas a través de una red de interconexión más rápida como conjunto de servidores conectados a través de redes rápidas.

• Memoria local (no hay problemas de consistencia).

• La interacción es sólo por pasaje de mensajes.

• Grado de acoplamiento de los procesadores:

• Multicomputadores (máquinas fuertemente acopladas). Procesadores y

red físicamente cerca. Pocas aplicaciones a la vez, cada una usando un

conjunto de procesadores.Alto ancho de banda y velocidad.

• Memoria compartida distribuida.

• Clusters.

• Redes (multiprocesador débilmente acoplado).

En este tipo de máquinas no existe la posibilidad de trabajar ÚNICAMENTE por memoria compartida, porque si están en máquinas separadas, no se podría acceder a la memoria de otra máquina. El tiempo de ejecución sería muy lento. No va a servir para programación paralela, si quiero reducir el tiempo, debería resolverlo por pasaje de mensajes o con programación híbrida que permita pasaje de mensajes y luego comunicación de mem distribuida.

**Evolución en respuesta a los cambios tecnológicos** → De enfoques ad-hoc iniciales a técnicas generales

60’s : Evolución de los SO. Más procesadores por chip para mayor potencia

de cómputo.

Controladores de dispositivos (canales) independientes permitiendo E/S → Interrupciones.

No determinismo. Multiprogramación. Problema de la sección crítica.

70’s: Formalización de la concurrencia en los lenguajes.

80’s: Redes, procesamiento distribuido.

90’s: MPP, Internet, C/S, Web computing.

2000’s: SDTR, computación móvil, Cluster y multicluster computing, sistemas colaborativos, computación pervasiva y ubicua, grid computing, virtualización.

Hoy: big data, IA, computación elástica, cloud computing, Green computing, bioinformática, redes de sensores, IoT, banca electrónica

**Clase 17/9**

concurrencia varios hilos.

cada proceso es un hilo de ejecución

si ver como de ejecutan y que imprime los if nuevos y do nuevos dijo que al de los teoricos lo puede preguntar.