**PROGRAMACION CONCURRENTE**

* **¿Qué es la concurrencia?**

Es la capacidad de ejecutar múltiples actividades en paralelo o simultáneamente. Permite a distintos objetos actuar al mismo tiempo.

* ¿Dónde encontramos concurrencia?

La concurrencia esta en todos lados, está presente en la naturaleza, la vida diaria, los sistemas de computo, etc. Cualquier sistema más o menos “inteligente” o complejo exhibe concurrencia, ejemplos como un Smartphone hasta un automóvil, un navegador web, accesos de varias aplicaciones a disco, juegos, etc.

* ¿Por qué es necesaria la Programación Concurrente?
  + Existen aplicaciones con estructura más natural:
    - El mundo no es secuencial.
    - Es más apropiado programas múltiples actividades independientes y concurrentes.
    - Reacción a entradas asincrónicas (Ej.: sensores en un STR).
  + Mejora en la respuesta:
    - No bloquear la aplicación completa por E/S.
    - Incremento en el rendimiento de la aplicación por mejor uso del hardware (ejecución paralela).
  + Sistemas distribuidos:
    - Una aplicación en varias máquinas.
    - Sistemas C/S o P2P.
* Objetivos de los sistemas concurrentes:
  + *Ajustar el modelo* de arquitectura de hardware y software al problema del mundo real a resolver. El mundo real es concurrente.
  + *Incrementar la performance*, mejorando los tiempos de respuesta de los sistemas de procesamiento de datos, a través de un enfoque diferente de la arquitectura física y lógica de las soluciones.
* Algunas ventajas:
  + Velocidad de ejecución que se puede alcanzar.
  + Mejor utilización de la CPU de cada procesador.
  + Explotación de la concurrencia inherente a la mayoría de los problemas reales.
* Concurrencia a nivel de Hardware
  + Multiprocesadores de memoria compartida:
    - La interacción se da modificando datos almacenados en la memoria compartida.
    - Esquemas UMA (acceso uniforme a memoria) con bus o crossbar switch (SMP, multiprocesadores simétricos). Problema de sincronización y consistencia.
    - Esquemas NUMA (acceso no uniforma a memoria) para mayor número de procesadores distribuidos. Problema de consistencia.
  + Multiprocesadores con memoria distribuida:
    - Procesadores conectados por una red. Cada uno tiene memoria local y la interaccion es solo por pasaje de mensajes.
    - Grado de acoplamiento de los procesadores:
      * Multicomputadores. Procesadores y red físicamente cerca. Pocas aplicaciones a la vez, cada una usando un conjunto de procesadores. Alto ancho de banda y velocidad.
      * Redes.
      * NOWs / Clusters.
      * Memoria compartida distribuida.
* **¿Qué es un proceso?**

Es un programa secuencial, y este es un solo flujo de control que ejecuta una instrucción y cuando esta finaliza ejecuta la siguiente.

Si tenemos un único THREAD o flujo de control, entonces es programación secuencial, monoprocesador.

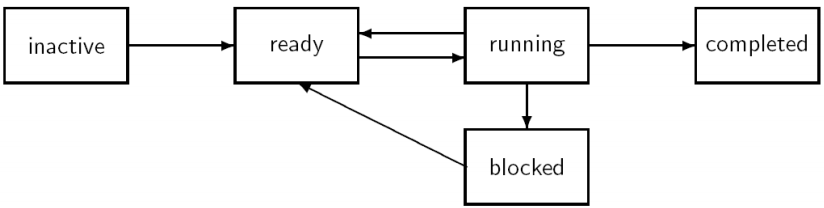
Si tenemos múltiples THREADS o flujos de control, entonces es programa concurrente, procesos paralelos.

Estos procesos cooperan y compiten.

La competencia es típica en SO y redes, debido a recursos compartidos (aparece el concepto de Deadlock y Starvation (inanición)).

En la cooperación los procesos se combinan para resolver una tarea común y es fundamental la sincronización.

* Procesamiento Secuencial:
  + Hay que fabricar un objeto compuesto por N partes o módulos y tenemos un monoprocesador (una maquina). La solución secuencial nos fuerza a establecer un estricto orden temporal. Al disponer de solo una maquina el ensamblado final del objeto solo se podrá realizar luego de N pasos de procesamiento o fabricación.
  + Un único programa que se ejecuta en un solo procesador.
* Procesamiento Paralelo:
  + Si disponemos de N máquinas para fabricar el objeto, y no hay dependencias, cada una puede trabajar al mismo tiempo en una parte.
    - Consecuencias: menor tiempo para completar el trabajo. Menor esfuerzo individual. Paralelismo del hardware.
    - Dificultades: distribución de la carga de trabajo. Necesidad de compartir recursos evitando conflictos. Necesidad de esperarse en puntos clave. Necesidad de comunicarse. Tratamiento de fallas. Asignación de una de las máquinas para el ensamblado.
* Procesamiento Concurrente:
  + Una maquina dedica parte del tiempo a cada componente del objeto 🡪 concurrencia sin paralelismo del hardware.
  + En un solo procesador se ejecutan muchos procesos secuenciales, alternándose en el tiempo.
    - Dificultades: distribución de carga de trabajo. Necesidad de compartir recursos evitando conflictos. Necesidad de esperarse en puntos clave. Necesidad de comunicarse. Necesidad de recuperar el “estado” de cada proceso al retomarlo.
* Multiprogramación en un procesador:
  + El tiempo de CPU es compartido entre varios procesos x por ejemplo time slicing.
  + El SO controla y planifica procesos: si el slice expiro o el proceso se bloque el SO hace context (process) switch.
  + Process Switch:
    - Suspender el proceso actual y restaurar otro.
      * Salvar el estado actual en memoria. Agregar el proceso al final de la cola de ready o una cola de wait.
      * Sacar un proceso de la cabeza de la cola ready. Restaurar su estado y ponerlo a correr.
    - Reanudar un proceso bloqueado:
      * Mover un proceso de la cola de wait a la de ready.
  + Cambios de estado de los procesos:



* **Programa Concurrente**

Un programa concurrente especifica dos o más programas secuenciales que pueden ejecutarse concurrentemente en el tiempo como tareas o procesos.

Un proceso o tarea es un elemento concurrente abstracto que puede ejecutarse simultáneamente con otros procesos o tareas (en paralelo), si el hardware lo permite.

Un programa concurrente puede tener N procesos habilitados para ejecutarse concurrentemente y un sistema concurrente puede disponer de M procesadores cada uno de los cuales puede ejecutar uno o más procesos.

Un programa concurrente es distinto a un sistema concurrente.

* + Características importantes:
    - Interacción.
    - No determinismo 🡪 dificultad para la interpretación y debug.
    - Ejecución “infinita”.
* Concurrencia vs Paralelismo
  + La concurrencia no es (solo) paralelismo.
  + Concurrencia “interleaved”:
    - Procesamiento simultaneo lógicamente.
    - Ejecución intercalada en un único procesador.
    - “Seudo-paralelismo”.
  + Concurrencia simultanea:
    - Procesamiento simultaneo físicamente.
    - Requiere un sistema multiprocesador o multicore.
    - Paralelismo “full”.
* Secuencialidad y Concurrencia:
  + Programa secuencial:
    - Totalmente ordenado.
    - Determinístico: para los mismos datos de entrada, ejecuta siempre la misma secuencia de instrucciones y obtiene la misma salida.
  + Los programas concurrentes puede ser NO determinísticos ya que pueden dar distintos resultados al ejecutarse sobre los mismos datos de entrada.
* Clases de aplicaciones
  + Programación concurrente:
    - Organizar software que consta de partes (relativamente) independientes.
    - Usar uno o múltiples procesadores.
  + Existen 3 grandes clases (superpuestas) de aplicaciones:
    - Sistemas multithreaded.
    - Sistemas de cómputo distribuido.
    - Sistemas de cómputo paralelo.
  + Multithreaded. Ejecución de N procesos independientes en M procesadores (N>M).
    - * Un sistema de software de “multithreading” maneja simultáneamente tareas independientes, asignando los procesadores de acuerdo a alguna política (ej, por tiempos). Organización más “natural” como un programa concurrente.
    - Ejemplos:
      * Sistemas de ventanas en PCs o WS.
      * Sistemas Operativos time-shared y multiprocesador.
      * Sistemas de tiempo real (por ej, en plantas industriales o medicina).
  + Computo distribuido
    - Una red de comunicaciones vincula procesadores diferentes sobre los que se ejecutan procesos que se comunican esencialmente por mensajes. Cada componente del sistema distribuido puede hacer a su vez multithreading.
    - Ejemplos:
      * Servidores de archivos en una red.
      * Sistemas de BD en bancos y aerolíneas (acceso a datos remotos).
      * Servidores Web distribuidos (acceso a datos remotos).
      * Sistemas corporativos que integran componentes de una empresa.
      * Sistemas fault-tolerant que incrementan la confiabilidad.
  + Procesamiento paralelo
    - Resolver un problema en el menor tiempo (o un problema más grande en aproximadamente el mismo tiempo) usando una arquitectura multiprocesador en la que se pueda distribuir la tarea global en tareas (¿independientes? ¿Interdependientes?) que puedan ejecutarse en distintos procesadores. Paralelismo de datos y paralelismo de procesos.
    - Ejemplos:
      * Calculo científico. Modelos de sistemas (meteorologías, movimiento planetario, etc.).
      * Gráficos, procesamiento de imágenes, efectos especiales, procesamiento de video, realidad virtual.
      * Problemas combinatorios y de optimización lineal o no lineal. Modelos econométricos.
* ¿Cómo mido el incremento de performance?

El procesamiento paralelo lleva a los conceptos de speedup y eficiencia.

* + Speedup:
    - S = Ts / Tp
  + Eficiencia:
    - E = S / p
* Conceptos básicos de concurrencia
  + Un programa concurrente puede ser ejecutado por:
    - Multiprogramación: los procesos comparten uno o más procesadores.
    - Multiprocesamiento: cada proceso corre en su propio procesador pero con memoria compartida.
    - Procesamiento distribuido: cada proceso corre en su propio procesador conectado a los otros a través de una red.
  + Los procesos se comunican:
    - La comunicación indica el modo en que se organiza y trasmiten datos entre tareas concurrentes. Esta organización requiere especificar protocolos para controlar el progreso y corrección de la comunicación.
    - Comunicación por memoria compartida:
      * Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella. Lógicamente no pueden operar simultáneamente sobre la MC, lo que obliga a bloquear y liberar el acceso a la memoria (ej: semáforos).
    - Comunicación por pasaje de mensajes:
      * Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos. También el lenguaje debe proveer un protocolo adecuado. Para que la comunicación sea efectiva los procesos deben “saber” cuando tienen mensajes para leer y cuando deben trasmitir mensajes.
  + Los procesos se sincronizan:
    - La sincronización es la posesión de información acerca de otro proceso para coordinar actividades.
    - El objetivo de la sincronización es restringir las historias de un programa concurrente solo a las permitidas.
    - Los procesos se sincronizan por exclusión mutua en el acceso a secciones críticas de código para no ejecutar simultáneamente, y por condición.
    - Ejemplos:
      * Sincronización para la reserva de pasajes.
      * Completar las escrituras antes de que comience una lectura.
      * Cajero: dar el dinero solo luego de haber verificado la tarjeta.
      * Compilar una clase antes de reanudar la ejecución.
      * No esperar por algo indefinidamente, si la otra parte esta “muerta”.
    - En la mayoría de los sistemas el tiempo absoluto no es importante. Con frecuencia los sistemas son actualizados con componentes más rápidas. La corrección no debe depender del tiempo absoluto.
    - Estado de un programa concurrente:
      * Cada proceso ejecuta un conjunto de sentencias, cada una implementada por una o más acciones atómicas (indivisibles, los estados intermedios son invisibles para los otros procesos).
    - Historia (trace) de un programa concurrente:
      * Es una ejecución dada por un intercalado (interleaving) particular de acciones individuales de los procesos.
      * El número posible de historias de un programa concurrente es generalmente enorme.
    - Un programa concurrente con n procesos, donde cada uno ejecuta m acciones atómicas tiene una cantidad de historias posibles dadas por (n\*m)! / (m!) elevado a n. Ejemplo, para 3 procesos con 2 acciones, hay 90 interleavings posibles.
    - Algunas historias son válidas y otras no. Se debe asegurar un orden temporal entre las acciones que ejecutan los procesos. Las tareas se intercalan en el tiempo 🡪 deben fijarse restricciones.
    - Formas de sincronización:
      * Sincronización por exclusión mutua: asegurar que solo un proceso tenga acceso a un recurso compartido en un instante de tiempo. Si el programa tiene secciones críticas que pueden compartir más de un proceso, EM evita que dos o más procesos puedan encontrarse en la misma sección crítica al mismo tiempo.
      * Sincronización por condición: permite bloquear la ejecución de un proceso hasta que se cumpla una condición dada.
      * Ejemplo de los dos mecanismos de sincronización en un problema de utilización de un área de memoria compartida: buffer limitado con productores y consumidores.
* Conceptos relacionas con PC:
  + Prioridad:
    - Un proceso que tiene mayor prioridad puede causar la suspensión (pre-emption) de otro proceso concurrente. Análogamente puede tomar un recurso compartido, obligando a retirarse a otro proceso que lo tenga en un instante dado.
  + Granularidad:
    - Elección de una granularidad: para una dada aplicación, significa optimizar la relación entre el número de procesadores y el tamaño de memoria total.
    - Grano fino y grano grueso.
    - Puede verse también como la relación entre cómputo y comunicación.
  + Manejo de los recursos:
    - Uno de los temas principales de la programación concurrente es la administración de los recursos compartidos. Esto incluye la asignación de recursos compartidos, métodos de acceso a los recursos, bloque 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).
  + El problema del 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.
    - Propiedad necesarias y suficientes para que exista deadlock:
      * Recursos reusables serialmente: los procesos comparten recursos que puede 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 solo son liberados voluntariamente.
      * Espera cíclica: existe una cadena circular (ciclo) de procesos t.q. cada uno tiene un recurso que su sucesor en el ciclo está esperando adquirir.
* Algunos comentarios:
  + Posible reducción de performance por overhead de context switch, comunicación, sincronización, etc.
  + Mayor tiempo de desarrollo y puesta a punto respecto de los programas secuenciales, y puede aumentar el costo de los errores.
  + La paralelización de algoritmos secuenciales no es un procesos directo que resulte fácil de automatizar.
  + Para obtener una mejora real de performance, se requiere adaptar el software concurrente al hardware paralelo (mapeo).
* Requerimientos para un lenguaje concurrente:

Independientemente del mecanismo de comunicaciones/sincronización entre procesos, los lenguajes de programación concurrente deberán proveer primitivas adecuadas para la especificación e implementación de las mismas.

* + De un lenguaje de programación concurrente se requiere:
    - Indicar las tareas o procesos que pueden ejecutarse concurrentemente.
    - Mecanismos de sincronización.
    - Mecanismos de comunicación entre los procesos.
* Resumen de conceptos:
  + La concurrencia es un concepto de software (propiedad del programa).
  + La programación paralela se asocia con la ejecución concurrente en múltiples procesadores que pueden tener memoria compartida, y con un objetivo de incrementar performance (propiedad de la maquina).
  + La programación distribuida es un “caso” de concurrencia con múltiples procesadores y sin memoria compartida.
  + En programación concurrente la organización de procesos y procesadores constituyen la arquitectura del sistema concurrente.
  + Especificar la concurrencia es esencialmente especificar los procesos concurrentes, su comunicación y sincronización.