**Teoría 1**

Objetivos de los sistemas concurrentes

* Ajustar el modelo de arquitectura de hardware y software al problema del mundo real a resolver
* Incrementar la performance, mejorando los tiempos de respuesta de los sistemas de cómputo, a través de un enfoque diferente de la arquitectura física y lógica de las soluciones.

**Ventajas**

* La 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**

Límite físico en la velocidad de los procesadores

* Máquinas monoprocesador ya no pueden mejorar
* Más procesadores por chip para mayor potencia de cómputo.
* Multicores → Cluster de multicores → Consumo.
* **Uso eficiente → Programación concurrente y paralela.**

Niveles de memoria.

* Jerarquía de memoria. ¿Consistencia?
* Diferencias de tamaño y tiempo de acceso.
* Localidad temporal y espacial de los datos

Multiprocesadores de memoria compartida.

* Interacción modificando datos en la memoria compartida.
* Esquemas UMA con bus o crossbar switch (SMP, multiprocesadores simétricos). Problemas de sincronización y consistencia.
* Esquemas NUMA para mayor número de procesadores distribuidos.
* Problema de consistencia

Multiprocesadores con memoria distribuida.

* Procesadores conectados por una red.
* Memoria local (no hay problemas de consistencia).
* 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)

**¿Qué es un proceso?**

**Programa Secuencial:** un único flujo de control que ejecuta una instrucción y cuando esta finaliza ejecuta la siguiente.  
**Proceso:** programa secuencial.

Un único hilo o flujo de control

* programación secuencial, monoprocesador

Múltiples hilos o flujos de control

* programa concurrente
* procesos paralelos

**Posibles comportamientos de los procesos**

* Procesos independientes: Relativamente raros. Poco interesante.
* Competencia: Típico en Sistemas Operativos y Redes, debido a recursos compartidos.
* Cooperación: Los procesos se combinan para resolver una tarea común. Sincronización.

**Procesamiento secuencial, concurrente y paralelo**Analicemos la solución ***secuencial*** y monoprocesador (***una máquina***) para fabricar un objeto compuesto por N partes o módulos. La solución secuencial nos fuerza a establecer un estricto orden temporal.  
Al disponer de sólo una máquina, el ensamblado final del objeto se podrá realizar luego de N pasos de procesamiento (la fabricación de cada parte).  
Si disponemos de ***N máquinas*** para fabricar el objeto, y no hay dependencia (por ejemplo de la materia prima), cada una puede trabajar ***al mismo tiempo*** en una parte. ***Solución Paralela.***

**Consecuencias**

* Menor tiempo para completar el trabajo.
* Menor esfuerzo individual.
* Paralelismo del hardware.

**Dificultades**

* Distribución de la carga de trabajo (diferente tamaño o tiempo de fabricación de cada parte, diferentes especializaciones de cada máquina y/o velocidades).
* Necesidad de compartir recursos evitando conflictos.
* Necesidad de esperarse en puntos clave.
* Necesidad de comunicarse.
* Tratamiento de las fallas.

**Otro enfoque: *una sola 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 al retomarlo.

|  |
| --- |
| **Concurrencia:**  ***Concepto de software no restringido a una arquitectura particular de hardware ni a un número determinado de procesadores.*** |

**Este último caso sería multiprogramación en un procesador**

* El tiempo de CPU es compartido entre varios procesos, por ejemplo por ***time slicing***.
* El sistema operativo controla y planifica procesos: si el slice expiró o el proceso se bloquea el sistema operativo hace ***context (process) switch***

***Process switch: suspender el proceso actual y restaurar otro***

1. Salvar el estado actual en memoria. Agregar el proceso al final de la cola de ***ready*** o una cola de ***wait***.
2. 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.***

|  |
| --- |
| **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, si el hardware lo permite (por ejemplo los TASKs de ADA).  
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.

**Características importantes:**

* Interacción
* No determinismo ⇒ dificultad para la interpretación y debug

**La concurrencia no implica paralelismo**

|  |  |
| --- | --- |
| **Concurrencia** “***Interleaved***”   * Procesamiento simultáneo lógicamente. * Ejecución intercalada en un único procesador. * Pseudo-paralelismo. | **Concurrencia** “***Simultánea***”   * Procesamiento simultáneo físicamente. * Requiere un sistema multiprocesador o multicore. * Paralelismo “full”. |

**Procesos e Hilos**

* Todos los Sistemas Operativos soportan ***Procesos:*** Cada proceso tiene su propio espacio de direcciones y recursos.
* Algunos Sistemas Operativos soportan procesos livianos ***(threads o hilos).***
* Proceso “liviano” que tiene su propio contador de programa y su pila de ejecución, pero no controla el “contexto pesado” (por ejemplo, las tablas de página).
* Todos los hilos de un proceso comparten el mismo espacio de direcciones y recursos.
* El programador o el lenguaje deben proporcionar mecanismos para evitar interferencias.
* La concurrencia puede estar provista por el lenguaje (ADA, Java) o por el Sistema Operativo (C/POSIX)

**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). |

**Programa concurrentes**

* **Ejemplo**

**x=0; //P** Instrucciones lógicamente concurrentes → Orden de ejecución es irrelevante  **y=0; //Q  
z=0; //R**

P → Q → R  
Q → P → R Darán el mismo resultado  
R → Q → P

* **Ejemplo.** Suponemos que *x = 5.* ***x = 0;* //P** Orden → Diferentes ejecuciones, pueden dar distintos resultados. ***x = x + 1;* //Q**  
  P → Q ⇒ *x = 1*

Q → P ⇒ *x = 0*

|  |
| --- |
| Los programas concurrentes pueden ser **no-determinísticos**: 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.

**Clases de aplicaciones**

|  |  |
| --- | --- |
| **Sistemas multithreaded**  Un sistema de software **“multithreading”** maneja simultáneamente múltiples actividades independientes, asignando los procesadores de acuerdo a alguna política. *Ejecución de N procesos independientes en M procesadores (N>M).*  **Ejemplo**: Sistemas de tiempo real (por ejemplo, en plantas industriales o medicina) | **Cómputo 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  **Ejemplo**: Servidores de archivos en una red; Sistemas de BD en bancos y aerolíneas (acceso a datos remotos) |

|  |
| --- |
| **Procesamiento paralelo** Resolver un problema en el menor tiempo (o un problema más grande en el mismo tiempo) usando una arquitectura multiprocesador en la que se pueda distribuir la tarea global en tareas que puedan ejecutarse en distintos procesadores. Paralelismo de datos y paralelismo de procesos.  **Ejemplo**: Cálculo científico para modelar y simular sistemas naturales |

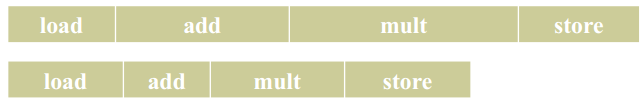
**Conceptos básicos de concurrencia**

* **Comunicación entre procesos**: La comunicación entre procesos concurrentes indica el modo en que se organiza y trasmiten datos entre tareas concurrentes. Esta organización requiere especificar ***protocolos*** para controlar el progreso y la corrección. Los procesos se comunican por *Memoria Compartida* o por *Pasaje de Mensajes:*

|  |  |
| --- | --- |
| **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 memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria. La solución más elemental es una variable de control tipo “semáforo” que habilite o no el acceso de un proceso a la memoria compartida. | **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” cuándo tienen mensajes para leer y cuando deben trasmitir mensajes. |

* **Sincronización entre procesos**: La sincronizaciónes la posesión de información acerca de otro proceso para  
  coordinar actividades. Los procesos se sincronizan por *exclusión mutua* o por *condición.*

|  |
| --- |
| ***El objetivo de la sincronización es restringir las historias de un programa concurrente sólo a las permitidas.*** |

* Ejemplos de cuando se debe sincronizar: Completar las escrituras antes de comenzar una lectura; Cajero: dar el dinero sólo luego de haber verificado la tarjeta.
* 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.
* El tiempo se ignora, sólo las secuencias son importantes
* Puede haber distintos ordenes (*interleavings*) en que se ejecutan las instrucciones de los diferentes procesos; los programas deben ser correctos para todos ellos
* *Una* ***acción atómica*** es una acción que hace una transformación de estado indivisible (estados intermedios invisibles para otros procesos).
* Cada proceso ejecuta un conjunto de sentencias, cada una implementada por una o más acciones atómicas.
* 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

|  |
| --- |
| ***Interferencia*:**  Un proceso toma una acción que invalida las suposiciones hechas por otro proceso |

Los procesos se sincronizan por *exclusión mutua* o por *condición.*

|  |  |
| --- | --- |
| **Sincronización por exclusión mutua**  Asegurar que sólo 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, la exclusión mutua 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. |

* **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 y grano grueso. |

* **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***.
* **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:

1. Recursos reusables serialmente: los procesos comparten recursos que pueden usar con exclusión mutua.
2. Adquisición incremental: los procesos mantienen los recursos que poseen mientras esperar adquirir recursos adicionales.
3. No-preemption: una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente.
4. 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 para un lenguaje concurrente
* Indicar las tareas o procesos que pueden ejecutarse concurrentemente.
* Mecanismos de sincronización.
* Mecanismos de comunicación entre los procesos.

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

**Problemas asociados con la Programación 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 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.

RESUMEN DE CONCEPTOS

|  |
| --- |
| * La **Concurrencia** es un concepto de software. * La **Programación Paralela** se asocia con la ejecución concurrente en múltiples procesadores que pueden tener memoria compartida, y con un objetivo de incrementar la performance (reducir el tiempo de ejecución). * La **Programación Distribuida** es un “caso” de concurrencia con múltiples procesadores. * 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*** |

**Teoría 2**

**Clases de instrucciones - Programación secuencial y concurrente**Un programa concurrente está formado por un conjunto de programas secuenciales.  
La programación secuencial estructurada puede expresarse con 3 clases de instrucciones básicas: **asignación**, **alternativa** (decisión) e **iteración** (repetición con condición). Se requiere una clase de instrucción para representar la concurrencia

|  |
| --- |
| **DECLARACIONES DE VARIABLES**  **Variable simple**  tipo variable = valor . Ej: int x = 8; int z, y;  **Arreglos**  int a[10]; int c[3:10] int b[10] = ([10] 2) int aa[5,5]; int cc[3:10,2:9] int bb[5,5] = ([5] ([5] 2)) |

|  |
| --- |
| **ASIGNACION**  **Asignación simple**: x = e  **Sentencia de asignación compuesta**: x = x + 1; y = y – 1; z = x + y a[3] = 6; aa[2,5] = a[4]  **Llamado a funciones**: x = f(y) + g(6) – 7  **swap**: v1 :=: v2  **skip**: termina inmediatamente y no tiene efecto sobre ninguna variable de programa. |
| **ALTERNATIVA**  **Sentencias de alternativa simple:** if B → SB expresión booleana. S instrucción simple o compuesta ({}). B “guarda” a S pues S no se ejecuta si B no es verdadera.  **Sentencias de alternativa múltiple:** if B1 → S1 B2 → S2 ..... Bn → Sn fiLas guardas se evalúan en algún orden arbitrario. Elección no determinística. Si ninguna guarda es verdadera el *if* no tiene efecto.  **Otra opción:** if (cond) S; if (cond) S1 else S2; |

|  |
| --- |
| **ITERACIÓN**  **Sentencias de alternativa ITERATIVA múltiple:** do B1 → S1 B2 → S2 ..... Bn → Sn od Las sentencias guardadas son evaluadas y ejecutadas hasta que todas las guardas sean falsas. La elección es no determinística si más de una guarda es verdadera.  **For-all: forma general de repetición e iteración** fa cuantificadores → Secuencia de Instrucciones af Cuantificador ≡ variable := exp\_inicial to exp\_final st B El cuerpo del *fa* se ejecuta 1 vez por cada combinación de valores de las variables de iteración. Si hay cláusula *such-that (st)*, la variable de iteración toma sólo los valores para los que *B* es true. Ejemplo: fa i := 1 to n, j := i+1 to n st a[i] > a[j] → a[i] :=: a[j] af  **Otra opción:** while (cond) S; for [i = 1 to n, j = 1 to n st (j mod 2 = 0)] S; |