Introducción a la PC

**Concurrencia** es la capacidad de ejecutar múltiples actividades en paralelo simultáneamente. Es relevante para el diseño del hardware, sistemas operativos, multiprocesadores, etc.

¿Por qué es necesaria la PC?

* No hay más ciclos de reloj, esto se debe a que con el multicore para ejecutar cosas más rápido tiene que ser concurrente, no alcanza con conseguir un mejor procesador.
* Son aplicaciones con estructura más natural, esto quiere decir que es más apropiado programar muchas actividades concurrentes e independientes y que muchas veces un problema es más fácil pensarlo de forma concurrente que secuencial (debido a que el mundo no funciona de forma secuencial).
* Hay una mejora en la respuesta, ya que no se bloquea la aplicación completa por E/S y hay un rendimiento mayor por un mejor uso del hardware (por la ejecución paralela).
* Los sistemas distribuidos necesitan la concurrencia para funcionar bien, ya que comparten recursos entre varias máquinas.

Objetivos

Los objetivos de un sistema concurrente son **ajustar el modelo** de arquitectura de hardware y software al problema del mundo real a resolver (el mundo no es secuencial) e **incrementar la performance,** mejorando los tiempos de respuesta de los sistemas de cómputo.

Como ventajas hay una mayor velocidad de ejecución, un mejor uso del CPU para cada procesador y se explota la concurrencia que ya existe en la mayoría de los problemas reales.

Comportamientos de los procesos

**Programa secuencial:** Un flujo de control que ejecuta una instrucción y cuando esta finaliza ejecuta la siguiente.

Los procesos pueden ser independientes, pueden cooperar o pueden competir (están compitiendo por usar un mismo recurso)

Los independientes son relativamente raros y no son interesantes para la materia. La competencia se suele dar en SO y Redes, porque comparten recursos.

Con el comportamiento de competencia aparece el **deadlock**, que es que los procesos se queden esperando a que el otro proceso libere el recurso compartido mientras que el otro está esperando un recurso del otro proceso (yo te doy la lapicera si me das la regla y vos me das la regla si te doy la lapicera).

También puede producirse **inanición**, ya que un proceso nunca puede agarrar el recurso compartido y se queda esperando.

El último comportamiento es la cooperación donde los procesos se combinan para resolver una tarea común. Involucra tomar un montón de decisiones.

En la programación concurrente los procesos se comportan de las tres maneras

Procesamiento secuencial concurrente y paralelos

Problema: Fabricar un objeto con varios componentes.

La solución **secuencial** nos fuerza a establecer un estricto orden temporal.

Una solución **paralela** (N máquinas) nos da menor tiempo para completar el trabajo, un menor esfuerzo individual y paralelismo del hardware, pero necesitamos distribuir la carga de trabajo, compartir los recursos evitando conflictos, comunicarse, tratar las fallas y ver que maquina se encarga del ensamblaje.

Otro enfoque es hacerlo **concurrente sin paralelismo**, esto quiere decir que uso una sola máquina que dedica parte del tiempo a cada componente del objeto. Sigue estando el problema de compartir recursos, distribuir la carga de trabajo, etc pero en menor medida ya que es una sola máquina. Se obtiene un mejor rendimiento, pero no tanto como en la solución paralela.

Concurrencia en sí

La **Concurrencia** es un concepto de software, no está restringido a una arquitectura de hardware particular ni a un número determinado de procesadores. Es dividir el problema en un conjunto de procesos que cooperan y compiten para resolverlo.

Un ejemplo de concurrencia sin paralelismo es la multiprogramación en un procesador, donde el CPU se comparte entre varios procesos y el SO controla y planifica los procesos. Si un proceso está mucho tiempo lo bloquea y pone en espera, y ejecuta un tiempo otro proceso, moviéndose a la cola de wait y ready según convenga.

Programa concurrente

Un **Programa Concurrente**  son dos o más programas secuenciales que pueden ejecutarse concurrentemente en el tiempo como tareas o procesos:

* Las tareas o procesos son elementos concurrentes abstractos que se pueden ejecutar simultáneamente con otros procesos o tareas si el hardware lo permite.
* Puede haber N procesos habilitados para ejecutarse a la vez y el sistema puede tener M procesadores que pueden ejecutar uno o más procesos.
* Se caracterizan por su interacción y el no determinismo, este último puede hacer difícil la interpretación y debug.

Procesos e hilos

**Proceso:** Cada proceso tiene su propio espacio de direcciones y recursos

**Proceso liviano o hilo:**

* Tiene su propio contador de programa y pila de ejecución, pero no controla el contexto pesado.
* Todos los hilos de un proceso comparten el mismo espacio de direcciones y recursos.
* El programador o el lenguaje deben proporcionar formas de evitar interferencias.
* La concurrencia puede estar provista por el lenguaje o por el sistema operativo.

Conceptos básicos de concurrencia

Comunicación entre procesos

La comunicación entre procesos indica cómo se organizan y transmiten datos entre tareas concurrentes. Requiere especificar protocolos para controlar el progreso y la corrección.

Se pueden comunicar por **memoria compartida** o **pasaje de mensajes.**

En **memoria compartida** los procesos intercambian información sobre la memoria o actúan sobre datos que están en la misma. No pueden operar al mismo tiempo sobre la memoria compartida, así que hay que bloquear y liberar el acceso a la memoria (para que no haya kilombo con el dato). Usualmente se usan **semáforos** que habiliten o no el acceso de un proceso a la memoria compartida.

En **pasaje de mensajes** hay que establecer un canal lógico o físico (en monoprocesador es lógico) para transmitir información, es la única forma de conectar a los procesos. El lenguaje debe tener un protocolo adecuado (ADA por ejemplo). Para que la comunicación funcione los procesos deben saber cuando tienen mensajes para leer y cuando los deben transmitir.

Sincronización entre procesos

La **sincronización** es la posesión de información acerca de otro proceso para coordinar actividades. Se sincronizan por **exclusión mutua** o por **condición**.

En **sincronización por exclusión mutua** nos aseguramos de que solo un proceso tenga acceso a un recurso a la vez. Si hay secciones críticas en el programa la exclusión mutua me garantiza que al menos un proceso va a estar en su sección crítica y evita que se encuentren en la misma al mismo tiempo.

En **sincronización por condición** podemos bloquear la ejecución de un proceso hasta que se cumpla una condición dada, el proceso espera hasta que pase algo (por ejemplo, que haya algo en un buffer o cola).

Interferencia

Un proceso toma una acción que invalida las suposiciones hechas por otro proceso. Por ejemplo, en una división por 0 chequeas que x sea distinto de 0, pero una vez que entraste te modifican x a 0 y se rompe todo, o te modifican la variable y te dejan trabajando con algo que es viejo.

Prioridad y granularidad

Un proceso con mayor **prioridad** puede causar que se suspenda otro proceso. Análogamente puede tomar un recurso compartido para obligar al proceso que lo estaba usando a salir y así otro proceso lo puede usar en ese momento.

La **granularidad** se da por la relación entre cómputo y comunicación. Si hay un programa con poco cómputo y comunicar se dice que es de **grano fino**, en cambio sí hay mucho código entre cada comunicación la solución sería de **grano grueso**. Hay que tratar de que la granularidad vaya acorde a la arquitectura usada. Por ejemplo, con mucho procesador conviene una solución de grano grueso, con poca comunicación para aprovecharlo al máximo.

Manejo de recursos

Un tema principal de la pc es la **administración de recursos compartidos**:

* Incluye la asignación de los recursos, métodos de acceso, bloqueo y liberación de recursos, seguridad y consistencia.
* Es una propiedad deseable que haya equilibrio en el acceso a recursos compartidos por todos los procesos, por eso se habla de **fairness**, que dice que todos los procesos deberían tener la posibilidad de acceder a un recurso compartido.
* Hay dos situaciones NO deseadas: **Inanición** (No logra acceder a los recursos compartidos) **y overloading** (la carga asignada excede su capacidad de procesamiento)
* También hay que evitar el **deadlock**

Problema de deadlock

**Cuatro propiedades necesarias y suficientes para que haya deadlock**

* **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 más. Se puede romper haciendo que agarren todo de una y no de a uno a la vez.
* **No-preemption:** Los recursos adquiridos por un proceso no pueden quitarse de manera forzada, sino que solo son liberados voluntariamente.
* **Espera cíclica:** Existe una cadena circular de procesos de forma que cada uno tiene un recurso que su sucesor en la cadena está esperando adquirir

Diagrama

Descripción generada automáticamente

Para que no haya deadlock al menos una no se tiene que dar nunca, con eso no vas a tener deadlock nunca más en tu programa.

Requerimientos para un lenguaje concurrente

Los lenguajes de PC tienen que dar herramientas para implementar la forma de comunicación o sincronización de procesos. Para eso deben indicar las tareas o procesos que se pueden ejecutar concurrentemente, los mecanismos de sincronización y los mecanismos de comunicación entre procesos.

Problemas de la PC

* Los procesos comparten recursos, no son independientes así que hay que añadir mecanismos de exclusión mutua y sincronización que agregan complejidad.
* Los procesos iniciados dentro un programa pueden dejar de existir en un momento sin
* que los otros procesos se enteren y quieran seguir comunicándose con ellos. Esto puede provocar deadlocks o una mala distribución de recursos.Hay un **no determinismo** implícito. Esto significa que dos ejecuciones de un mismo programa no necesariamente son idénticas, dificultando la interpretación y debug.
* Puede reducirse la performance por un overhead de context switch, comunicación, sincronización, etc.
* Mayor tiempo de desarrollo y puesta a punto, difícil paralelizar cosas secuenciales.
* Hay que adaptar el software al hardware para que haya una mejora real en el rendimiento.

Concurrencia vs paralelismo

La **concurrencia** no está restringida a una arquitectura o hardware particular. Implica especificar los procesos concurrentes, su comunicación y sincronización. Mientras que el **paralelismo** tiene que ver con ejecutar de forma concurrente algo en múltiples procesadores para reducir el tiempo de ejecución.

Diagrama

Descripción generada automáticamente

Concurrencia a nivel de hardware

Los procesadores se volvieron tan buenos que no podían seguir mejorando, así que se empezaron a poner más procesadores por chip para tener mayor potencia de cómputo. De ahí salieron los multicore, después el clúster de multicores. Salen a causa de esto nuevos paradigmas de comunicación ( comunicación híbrida ) y el problema de que tienen un mayor consumo energético.

El uso eficiente de estos monstruos se da con la programación concurrente y paralela.

Niveles de memoria

Hay una jerarquía de memoria, con un protocolo de coherencia de caché para que los datos compartidos sean consistentes y no haya incoherencias. Hay que considerar las diferencias de tamaño y tiempo de acceso a memoria. Además, se debe considerar la localidad temporal y espacial de los datos, es decir, cómo está guardada la información en memoria. Con una arquitectura multicore hay que aprovechar al máximo cada nivel de la jerarquía de memoria.

Las máquinas de **memoria compartida** (multicores) tienen una única memoria donde todas las CPU acceden a esta memoria. En cambio, las máquinas con **memoria distribuida** (clúster de multicores) cada CPU tiene su propia memoria y los datos entre memorias viajan por envío de mensajes.

Multiprocesadores de memoria compartida

* Interactúan modificando datos en la memoria compartida
* Pueden tener un esquema UMA donde todas las unidades de procesamiento tardan el mismo tiempo en acceder a la memoria compartida.
* Pueden tener un esquema NUMA para un mayor número de procesadores distribuidos, esto solo deja un problema de consistencia (revisar que cada memoria tenga todo bien)

Multiprocesadores de memoria distribuida

* Procesadores conectados por una red
* Cada uno tiene su memoria local, no hay problemas de consistencia
* Su interacción es solo por pasaje de mensajes
* No se podrían utilizar para una solución con paralelismo porque el tiempo de ejecución sería estúpidamente largo
* Distintos tipos de máquinas que difieren en su grado de acoplamiento, esto tiene que ver con cómo están conectadas. La comunicación es más lenta mientras menos grado de acoplamiento haya.