Clase 1

Concurrencia

Capacidad de ejecutar múltiples actividades en paralelo, permitiendo a distintos objetos actual al mismo tiempo.

No hay un orden preestablecido en la ejecución, dándole lugar al **no determinismo** y la concurrencia surge a partir de la necesidad de mejores rendimientos y procesadores más potentes.

Necesidad

- Aplicaciones con estructura más natural, debido a que intentamos representar problemas del mundo real el cual NO es secuencial.
- Mejora en la respuesta porque no hay bloqueo de la aplicación completa cuando hay una E/S además del incremento del rendimiento de la aplicación por mejor uso del hardware.
- Sistemas distribuidos.

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.

Comportamientos de los procesos

En un programa secuencial hay un único flujo de control que ejecuta una instrucción y cuando termina, ejecuta la próxima.

Desde ahora un PROCESO se asume que está ya cargado y ejecutándose y habrá

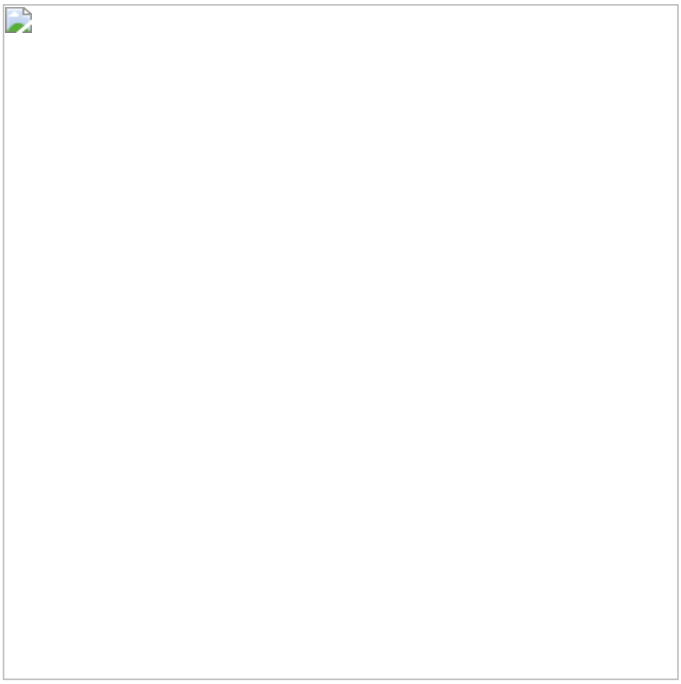
- Cooperación, para resolver una tarea común, dándole lugar a la sincronización.
- Competir para acceder a un recurso compartido. En esta competencia surgirá el concepto de
    - Deadlock: situación en la que dos o más procesos en ejecución se quedan en un estado en el que están esperando indefinidamente el uno al otro para liberar recursos o hacer alguna acción necesaria para continuar la ejecución.
    - Inanición: un proceso no puede completar su trabajo porque se le niega el acceso a los recursos que necesita.

Por otro lado tengo los procesos independientes que son raros y poco interesantes.

Estados de los procesos

El S.O controla y planifica procesos, si el proceso se bloquea, el sistema operativo hará un context switch el cual suspende el proceso actual y restaura otro, deberá

1. Salvar el estado actual en memoria, agregando el proceso al final de la cola de ready o de wait.
2. Sacar un proceso de la cola de ready, restaurar su estado y ejecutarlo.



Programa concurrente

Debe especificar al menos dos programas secuenciales que pueden ejecutarse concurrentemente en el tiempo.

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

Será importante la interacción y el no determinismo (barrera para 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 como las tablas de páginas. Todos los hilos comparten el mismo espacio de direcciones y recursos.

Otros conceptos

- Comunicación entre procesos: el tipo de comunicación indica cómo se organizan y transmiten los datos. Será necesario establecer protocolos para controlar el progreso y la corrección. Se usará
    - Memoria compartida: los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella. No pueden operar simultáneamente sobre la memoria, entonces se **bloquea** y **libera** el acceso a la memoria.

      Esta solución se conoce como semáforo porque habilita o no el acceso de un proceso a la memoria compartida.
    - Pasaje de mensajes: se establece un canal para transmitir información entre procesos. Los procesos deberán saber cuándo tienen mensajes para leer o para transmitir.
- Sincronización: posesión de información acerca de otro proceso para coordinar actividades, se hará mediante

    - Exclusión mutua: asegura 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, la exclusión mutua evita que dos o más procesos puedan encontrarse en la misma sección crítica al mismo tiempo.
    - Condición: permite bloquear la ejecución de un proceso hasta que se cumpla una condición dada.
- Interferencia: un proceso toma una acción que invalida las suposiciones hechas por otro proceso.

```
Ejemplo 1: nunca se debería dividir por 0.
int x, y, z;
process A1
{ ….
  y = 0;
….
}
process A2
{ …..
  if (y <> 0) z = x/y;
..…
}
```

- Prioridad: un proceso con mayor prioridad puede causar la suspensión de otro proceso concurrente. Podrá tomar un recurso compartido, obligando a retirarse a otro proceso que lo tenga en un instante dado.
- Granularidad de una aplicación: medida de la cantidad de trabajo computacional y comunicación entre hilos o procesos en una aplicación. Esencialmente, se trata de cuánto trabajo se realiza antes de que se requiera algún tipo de interacción o comunicación entre los elementos concurrentes. En esta se evalúa la complejidad del tipo de dato y el tamaño en el que se dividen los campos de datos.
    - Grano fino: el trabajo computacional se divide en tareas más pequeñas y cada tarea requiere poca cantidad de tiempo antes de que se necesite alguna comunicación o sincronización entre hilos o procesos. Podrían ser asignaciones o funciones simples individuales.
    - Grano grueso: el trabajo computacional se divide en tareas más grandes, lo que significa que cada tarea se ejecuta durante un período más prolongado antes de que se requiera alguna comunicación o sincronización. Por ejemplo clases complejas o módulos dentro de otros módulos.
- Administración de recursos compartidos: se asignan, se definen métodos de acceso a los recursos, bloqueo y liberación, seguridad y consistencia.

  Una propiedad deseable de los sistemas concurrentes es el equilibrio en el acceso a recursos compartidos.

  Dos situaciones NO deseadas será
    - Inanición.
    - Overloading: situación en la que se asigna o se sobrecarga un sistema o recurso con más trabajo del que puede manejar eficientemente.
    - Deadlock: dos o más procesos entrarán en deadlock si por error ambos quedan esperando que el otro libere un recurso compartido. Hay propiedades necesarias y suficientes para que exista deadlock

        - Recursos reusables serialmente: los procesos comparten recursos que pueden usar con exclusión mutua.
        - Adquisición incremental: los procesos mantienen los recursos que poseen mientras esperar adquirir recursos adicionales.
        - No-preemption: una vez que un proceso adquiere (o bloquea) ciertos recursos, estos recursos no pueden ser retirados o liberados por la fuerza por parte del sistema operativo o de otros procesos. En cambio, solo pueden ser liberados de manera voluntaria por el proceso que los posee.
        - 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.
- Lenguaje concurrente: deberán proveer instrucciones adecuadas para la especificación e implementación de las mismas. Hay ciertos requerimientos:
    - Indicar las tareas o procesos que pueden ejecutarse concurrentemente.
    - Mecanismos de sincronización.
    - Mecanismos de comunicación entre los procesos.

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. Liveness es la propiedad que garantiza que un sistema concurrente continúe avanzando y realizando progreso, incluso cuando existen condiciones potenciales de bloqueo o espera infinita.
- 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. Estos procesos pueden ejecutarse de manera intercalada y competir por recursos como la CPU, la memoria o dispositivos de entrada/salida.
- Posible reducción de performance por overhead de context switch, comunicación, sincronización. Overhead es el costo adicional asociado con la transición de un proceso o hilo en ejecución a otro. Es decir, es el tiempo y los recursos adicionales necesarios para realizar el cambio de contexto.
- Mayor tiempo de desarrollo y puesta a punto porque es difícil paralelizar algoritmos secuenciales.
- Necesidad de adaptar el software concurrente al hardware paralelo para mejora real en 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.

Paralelismo

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

Concurrencia a nivel de hardware

Hay un límite físico en la velocidad de los procesadores ya que

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

También se evalúa los niveles de memoria

- Memoria compartida: la interacción entre hilos o procesos que modifican datos en la memoria compartida es un aspecto crítico. Esta interacción se ve influenciada por la arquitectura del sistema, que puede ser de tipo UMA (Acceso Uniforme a Memoria) con bus o crossbar switch, como en sistemas SMP (Procesadores Simétricos Multiprocesador). En estos escenarios, surgen desafíos de sincronización y consistencia, ya que múltiples hilos pueden acceder a la memoria compartida simultáneamente, lo que requiere mecanismos adecuados para asegurar que los datos sean modificados de manera coherente y precisa.

  Además, en arquitecturas más avanzadas como NUMA (Acceso No Uniforme a Memoria), diseñadas para soportar un mayor número de procesadores distribuidos, se plantean cuestiones adicionales. La interacción entre procesadores puede provocar problemas de consistencia en la memoria compartida, ya que los datos almacenados en diferentes nodos NUMA pueden tardar más tiempo en sincronizarse y mantenerse consistentes.

- Memoria distribuida: os multiprocesadores con memoria distribuida representan una configuración donde varios procesadores están interconectados mediante una red. Cada procesador posee su propia memoria local, lo que evita problemas de consistencia de datos. En esta configuración, la interacción entre procesadores se logra exclusivamente a través de pasaje de mensajes, lo que garantiza una comunicación eficiente y controlada.

  El grado de acoplamiento entre los procesadores puede variar, dando lugar a diferentes arquitecturas. Por un lado, los multicomputadores, caracterizados por una fuerte vinculación, colocan procesadores y red en proximidad física. En este escenario, se ejecutan pocas aplicaciones simultáneamente, cada una aprovechando un conjunto de procesadores. Este enfoque brinda un alto ancho de banda y velocidad de comunicación, ideal para cargas de trabajo específicas.

  Por otro lado, se encuentra la memoria compartida distribuida, en la que los procesadores están conectados en clusters. Esta configuración se asemeja a una red de multiprocesador de acoplamiento más débil, en la que los procesadores interactúan a través de la red. Estos clusters permiten una mayor flexibilidad en la ejecución de aplicaciones y, a pesar del menor acoplamiento, aún pueden ofrecer ventajas significativas en términos de rendimiento y escalabilidad.