Programación Concurrente 2017

Clase 1

Facultad de Informática

UNLP

Metodología del curso

- **Comunicación:** Plataforma IDEAS (ideas.info.unlp.edu.ar).
  - Solicitar inscripción.

- **Bibliografía / material:**
  - ***Libro base:*** Foundations of Multithreaded, Parallel, and Distributed Programming. Gregory Andrews. Addison Wesley. (www.cs.arizona.edu/people/greg/mpdbook).
  - Material de lectura adicional: bibliografía, web.
    - Principles of Concurrent and Distributed Programming, 2/E. Ben-Ari. Addison-Wesley
    - An Introduction to Parallel Computing. Design and Analysis of Algorithms, 2/E. Grama, Gupta, Karypis, Kumar. Pearson Addison Wesley.
    - The little book of semaphores. Downey. http://www.cs.ucr.edu/~kishore/papers/semaphores.pdf.
  - Planteo de temas/ejercicios (recomendado hacerlos).

Metodología del curso

- **Horarios:**
  - Teoría (2 horarios):
    - Lunes de 8 a 11 - aula 5.
    - Jueves de 14 a 17 - aula 4.
  - Prácticas:
    - Martes de 8 a 11 - aula 11.
    - Jueves de 17:30 a 20:30 - aula 4.
    - Sábado de 9 a 12 - aula 1.
  - Explicación de Ejercicios:
    - Se hará en las clases de teoría.

Metodología del curso

- **Cursada:** la cursada se aprueba con un parcial que cuenta con dos recuperatorios.
- Para la promoción práctica (optativa) se deben rendir 3 parcialitos prácticos (en la teoría de los lunes):
  - Si los 3 parcialitos son aprobados con nota ≥ 8 → se obtiene la cursada.
  - Si el promedio de los parcialitos es ≥ 6 y al menos 2 son aprobados → rinde un parcial reducido (la condición vale tanto para el parcial como para los recuperatorios).
- **Final:** se aprueba por medio de un final tradicional teorico-práctico.
- Para la promoción teórica (optativa) se debe:
  - Rendir al menos 2 de 3 parcialitos teóricos (en la teoría de los lunes).
  - Una prueba teórica (en febrero de 2018). Dependiendo de la nota hay tiempo durante el semestre posterior para:
    - Nota ≥ 7 → coloquio en mesa de final.
    - Nota ≥ 4 y < 7 → trabajo individual.

Motivaciones del curso

- ¿Por qué es importante la concurrencia?
- ¿Cuáles son los problemas de concurrencia en los sistemas?
- ¿Cómo se resuelven usualmente esos problemas?
- ¿Cómo se resuelven los problemas de concurrencia a diferentes niveles (hardware, SO, lenguajes, aplicaciones)?
- ¿Cuáles son las herramientas?

Objetivos del curso

- Plantear los fundamentos de programación concurrente, estudiando sintaxis y semántica, así como herramientas y lenguajes para la resolución de programas concurrentes.

- Analizar el concepto de sistemas concurrentes que integran la arquitectura de Hardware, el Sistema Operativo y los algoritmos para la resolución de problemas concurrentes.

- Estudiar los conceptos fundamentales de comunicación y sincronización entre procesos, por Memoria Compartida y Pasaje de Mensajes.

- Vincular la concurrencia en software con los conceptos de procesamiento distribuido y paralelo, para lograr soluciones multiprocesador con algoritmos concurrentes.

Temas del curso

- **Conceptos básicos.** Concurrencia y arquitecturas de procesamiento. Multithreading, Procesamiento Distribuido, Procesamiento Paralelo.
- **Concurrencia por memoria compartida.** Procesos y sincronización. Locks y Barreras. Semáforos. Monitores. Resolución de problemas concurrentes con sincronización por MC.
- **Concurrencia por pasaje de mensajes (MP).** Mensajes asincrónicos. Mensajes sincrónicos. Remote Procedure Call (RPC). Rendezvous. Paradigmas de interacción entre procesos.
- **Lenguajes que soportan concurrencia.** Características. Similitudes y diferencias.
- **Introducción a la programación paralela.** Conceptos, herramientas de desarrollo, aplicaciones.

Concurrencia

¿Que es?

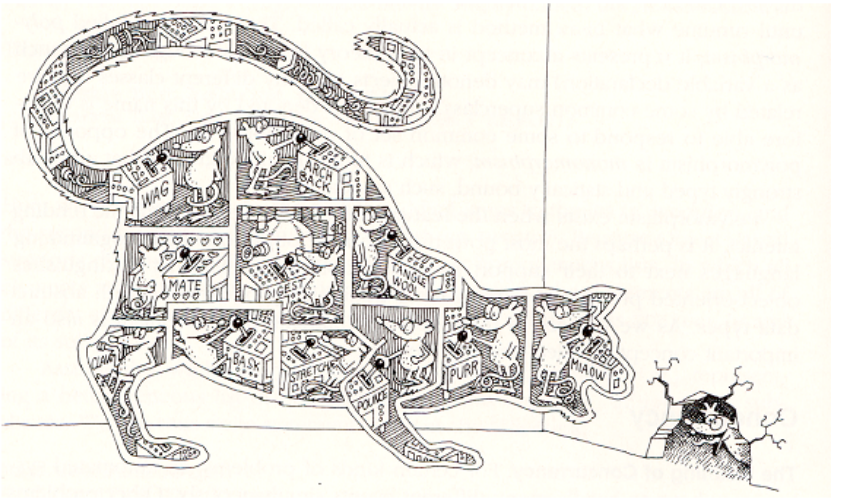
- Concurrencia es la capacidad de ejecutar múltiples actividades en paralelo o simultáneamente.
- Permite a distintos objetos actuar al mismo tiempo.
- Un concepto clave dentro de la ciencia de la computación. Factor relevante para el diseño de hardware, sistemas operativos, multiprocesadores, computación distribuida, programación y diseño.
- La necesidad de sistemas de cómputo cada vez más poderosos y flexibles atenta contra la simplificación de asunciones de secuencialidad.

¿Donde está?

- Navegador Web accediendo una página mientras atiende al usuario.
- Varios navegadores accediendo a la misma página.
- Acceso a disco mientras otras aplicaciones siguen funcionando.
- Impresión de un documento mientras se consulta.
- El teléfono avisa recepción de llamada mientras se habla.
- Varios usuarios conectados al mismo sistema (reserva de pasajes).
- Cualquier objeto más o menos “inteligente” exhibe concurrencia.
- Juegos, automóviles, etc.

Concurrencia

- Los sistemas biológicos suelen ser masivamente concurrentes: comprenden un gran número de células, evolucionando simultáneamente y realizando (independientemente) sus procesos.



- En el mundo biológico los sistemas secuenciales rara vez se encuentran.
- En algunos casos se tiende a pensar en sistemas secuenciales en lugar de concurrentes para simplificar el proceso de diseño. Pero esto va en contra de la necesidad de sistemas de cómputo cada vez más poderosos y flexibles.

Concurrencia “natural”

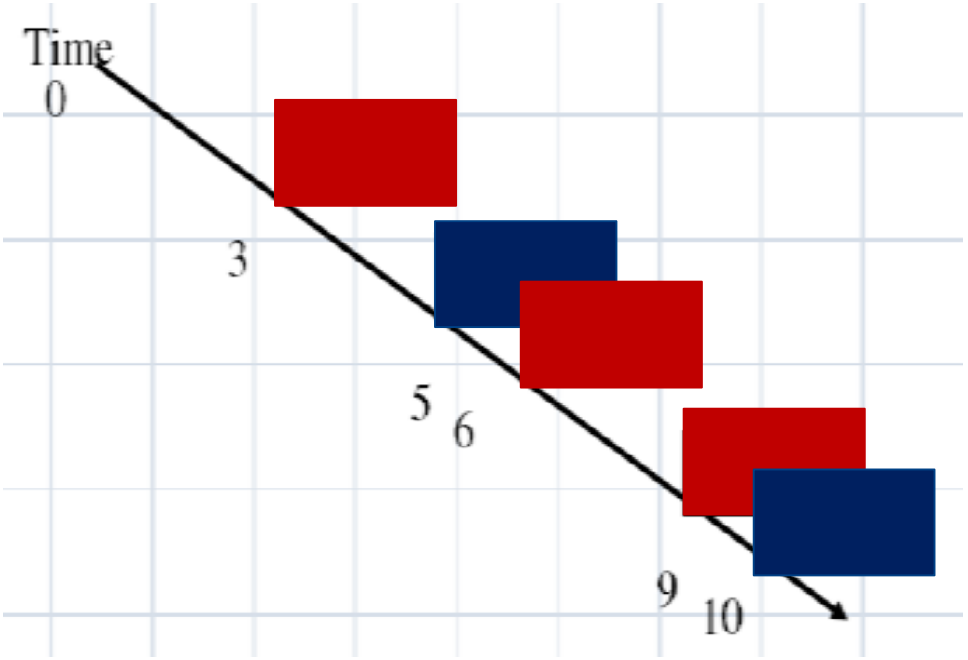
- **Problema:** Desplegar cada 3 segundos un cartel ROJO.

- **Solución secuencial:**

Programa Cartel
Mientras (true)
Demorar (3 seg)
Desplegar cartel
Fin mientras
Fin programa

Concurrencia “natural”

- **Problema:** Desplegar cada 3 segundos un cartel ROJO y cada 5 segundos un cartel AZUL.



Concurrencia “natural”

```
Programa Carteles
    Proximo_Rojo = 3
    Proximo_Azul = 5
    Actual = 0
    Mientras (true)
        Si (Proximo_Rojo < Proximo_Azul)
                Demorar (Proximo_Rojo – Actual)
                Desplegar cartel ROJO
                Actual = Proximo_Rojo
                Proximo_Rojo = Proximo_Rojo +3
        sino
                Demorar (Proximo_Azul – Actual)
                Desplegar cartel AZUL
                Actual = Proximo_Azul
                Proximo_Azul = Proximo_Azul +5
    Fin mientras
Fin programa
```

Concurrencia “natural”

- Obliga a establecer un orden en el despliegue de cada cartel.
- Código más complejo de desarrollar y mantener.
- ¿Que pasa si se tienen más de dos carteles?
- ***Más natural:*** cada cartel es un elemento independiente que actúa concurrentemente con otros → ***es decir, ejecutar dos o más algoritmos simples concurrentemente***.

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

- No hay un orden preestablecido en la ejecución

¿Por qué es necesaria la Programación Concurrente?

- No hay más ciclos de reloj → Multicore → ¿por qué? y ¿para qué?

- Aplicaciones con estructura más natural.
  - El mundo no es secuencial.
  - Más apropiado programar 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

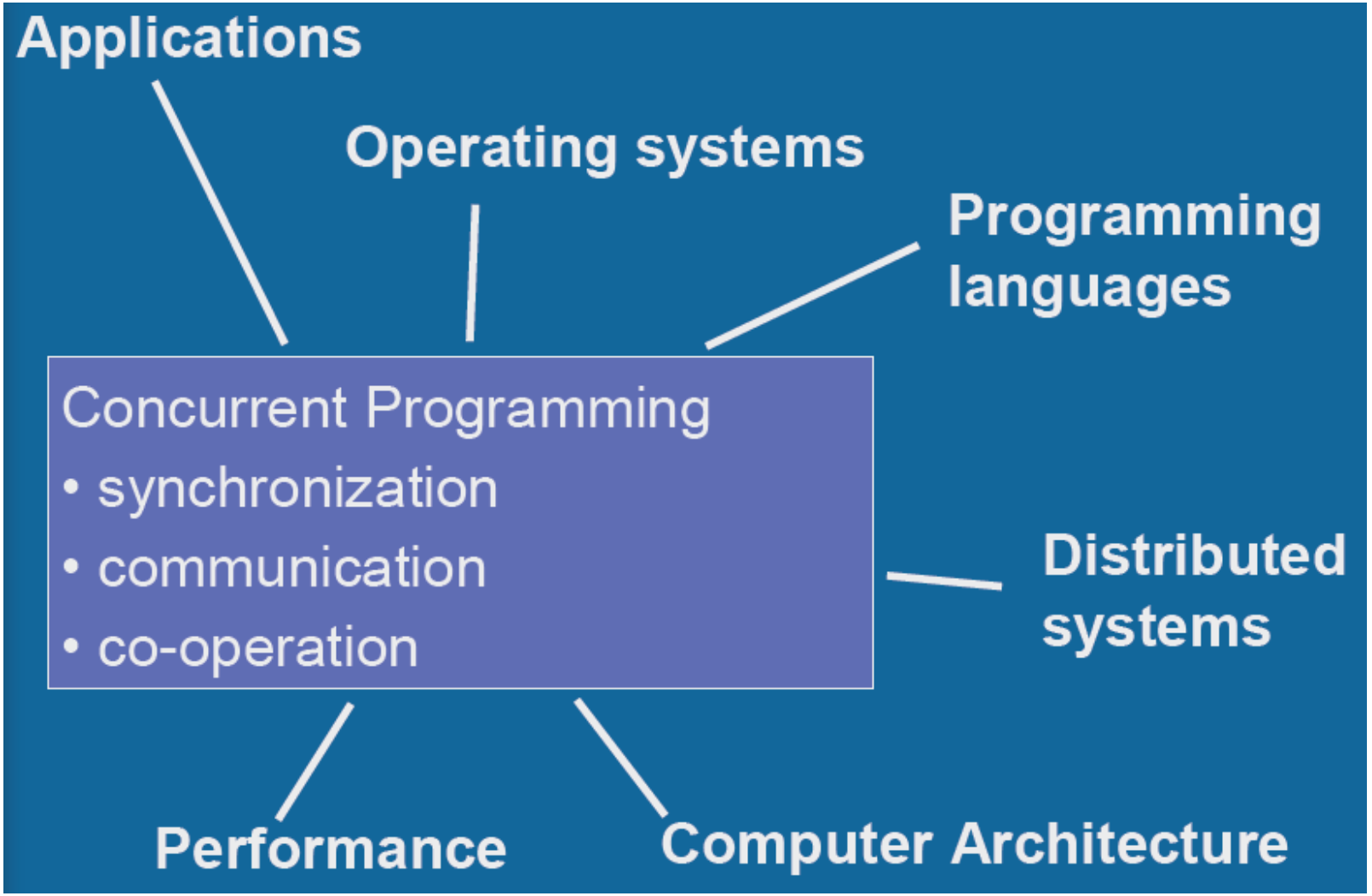
***Ajustar el modelo* de arquitectura de hardware y software al problema del mundo real a resolver.**

***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.**

Algunas 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.

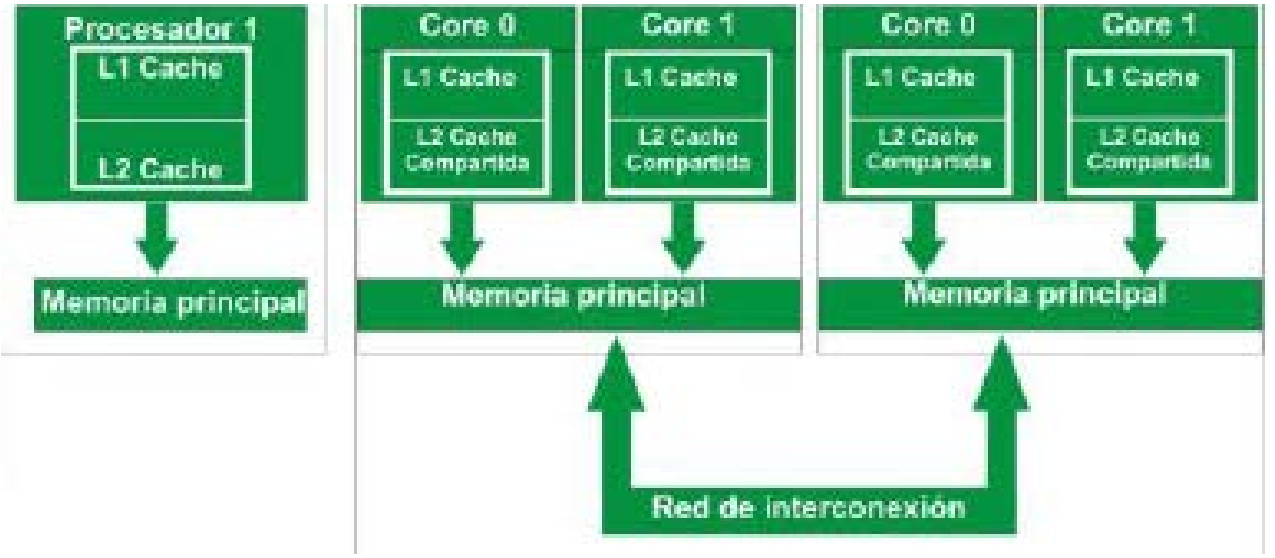
Conexiones


Applications
Operating systems
Programming languages
Concurrent Programming
• synchronization
• communication
• co-operation
Distributed systems
Performance
Computer Architecture

Concurrencia a nivel de hardware

Límite físico en la velocidad de los procesadores

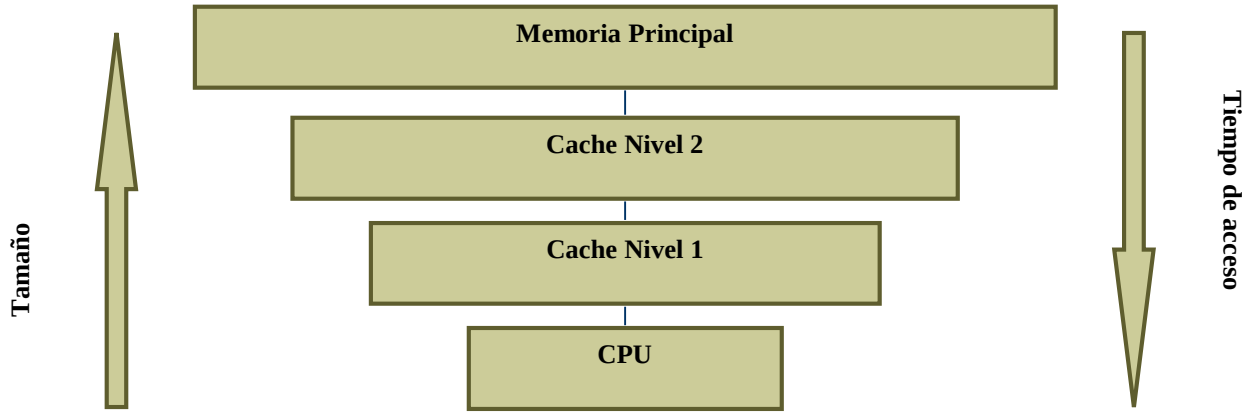
- Máquinas monoprocesador ya no pueden mejorar.
- Más procesadores por chip para mayor potencia de cómputo.
- Multicores → Cluster de multicores → Consumo.
- **Uso eficiente → Programación concurrente y paralela.**



Concurrencia a nivel de hardware

Niveles de memoria.

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

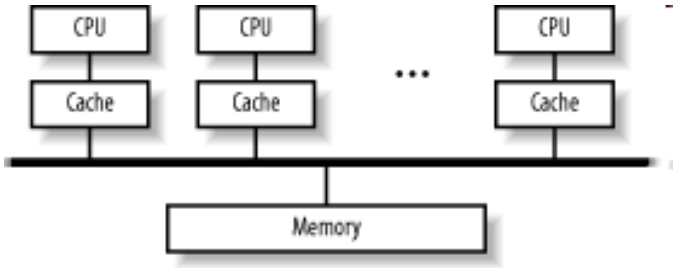


Máquinas de memoria compartida vs memoria distribuida.

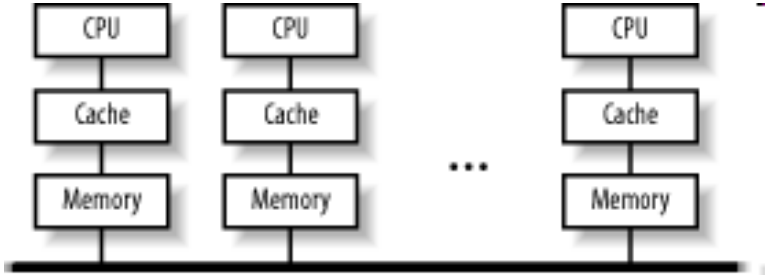
Concurrencia a nivel de hardware

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.



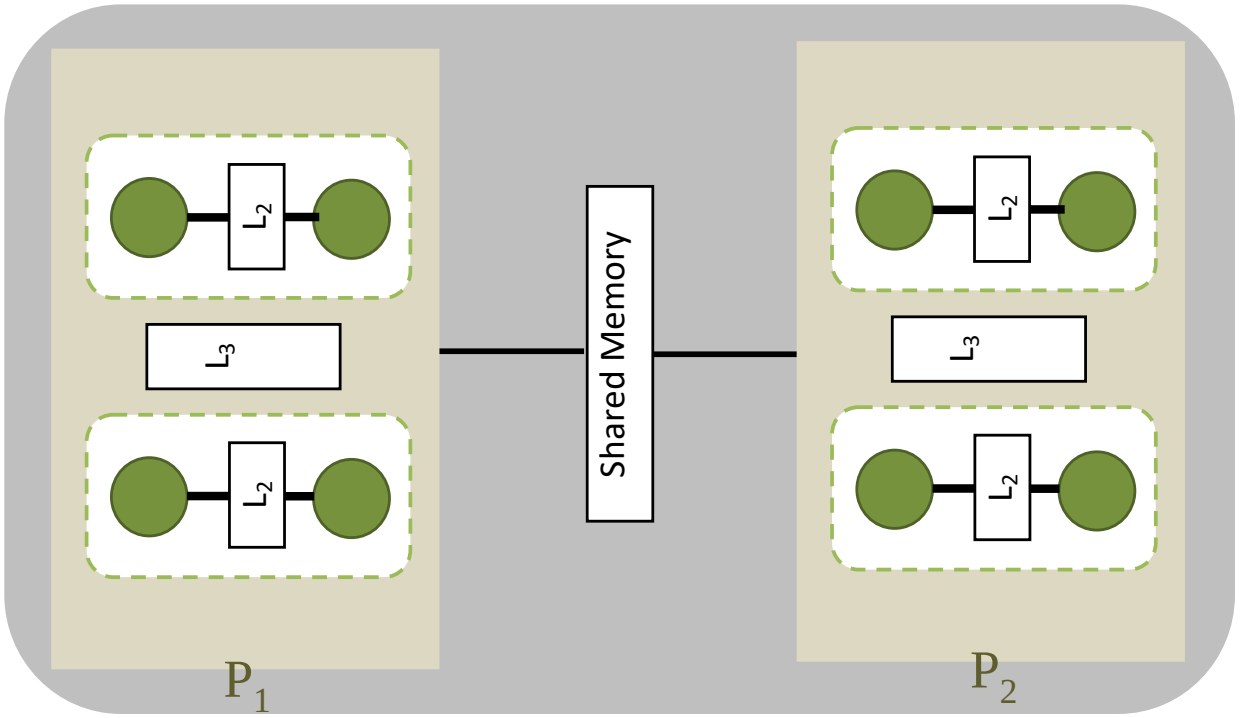
Esquema UMA



Esquema NUMA

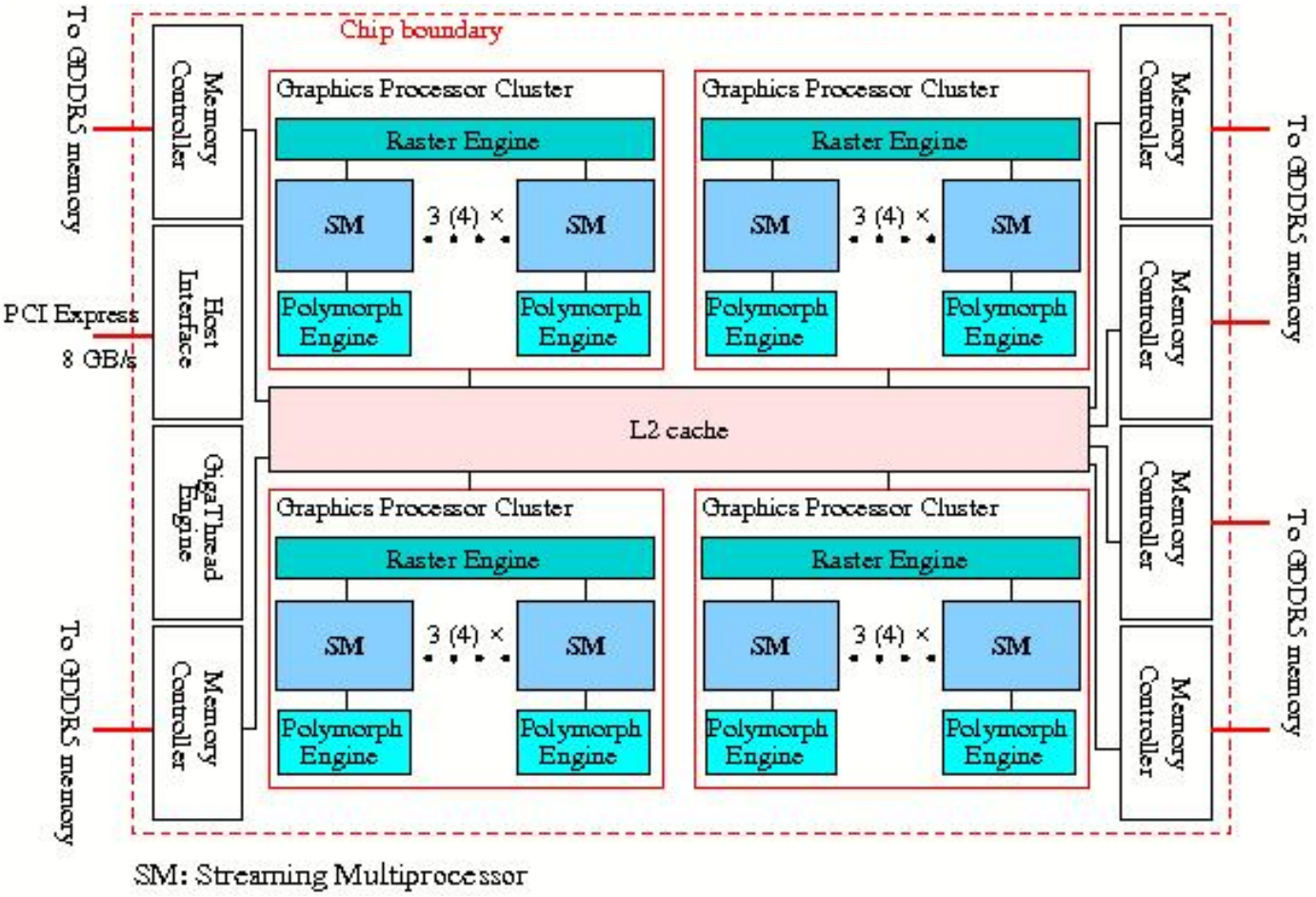
Concurrencia a nivel de hardware

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



Concurrencia a nivel de hardware

- Ejemplo de multiprocesador de memoria compartida: *GPU*.



Concurrencia a nivel de hardware

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).

¿Que 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.

Los procesos cooperan y compiten...

Posibles comportamientos de los procesos

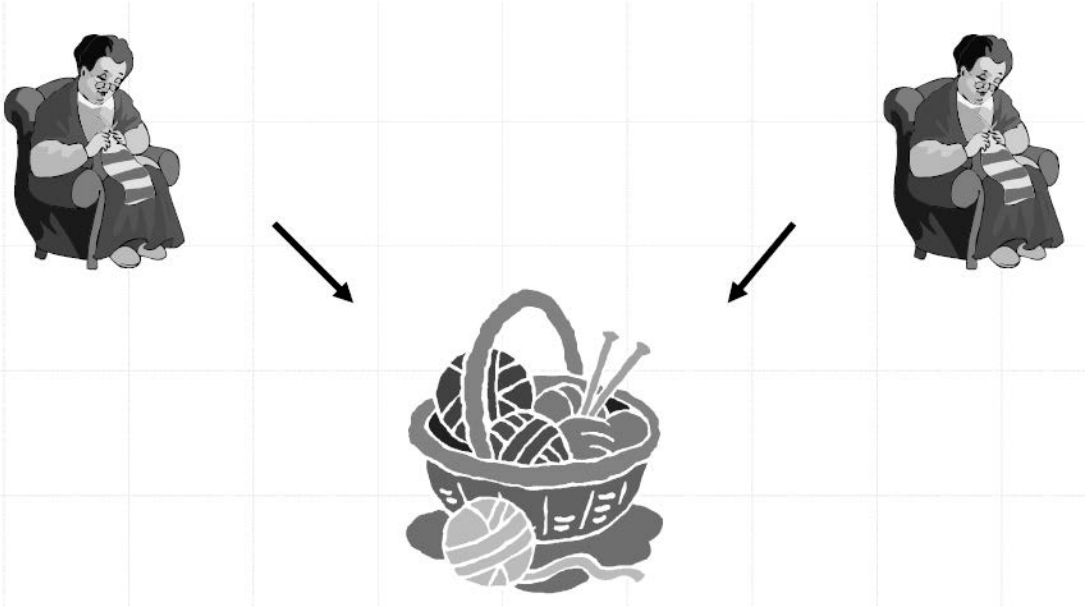
Procesos independientes

- Relativamente raros.
- Poco interesante.

Posibles comportamientos de los procesos

Competencia

- Típico en Sistemas Operativos y Redes, debido a recursos compartidos.

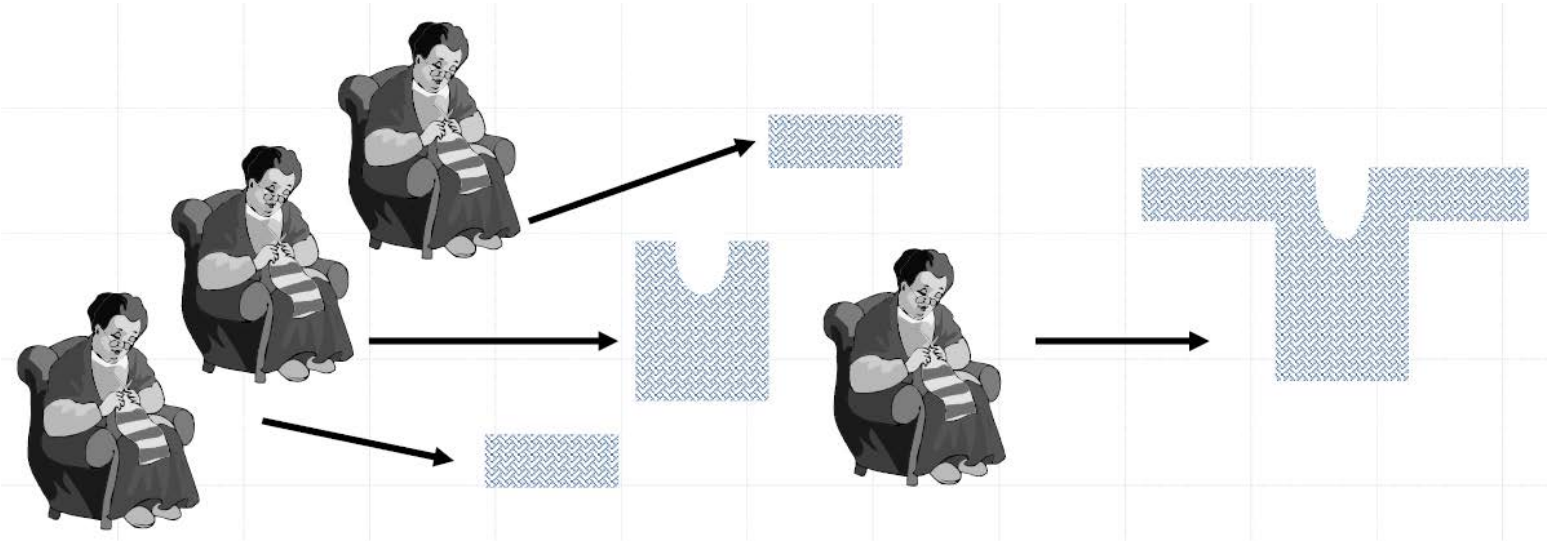


Deadlock Inanición

Posibles comportamientos de los procesos

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).

Procesamiento secuencial, concurrente y paralelo

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 ⇒

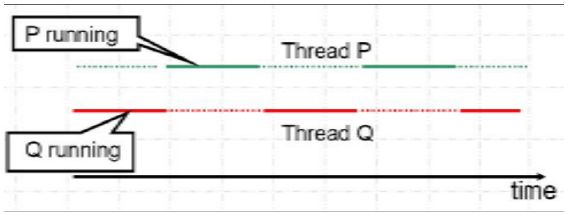
- Menor tiempo para completar el trabajo.
- Menor esfuerzo individual.
- Paralelismo del hardware.

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.
- Asignación de una de las máquinas para el ensamblado (¿Cual?).

Procesamiento secuencial, concurrente y paralelo

Otro enfoque: ***un sóla 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.

Procesamiento secuencial, concurrente y paralelo

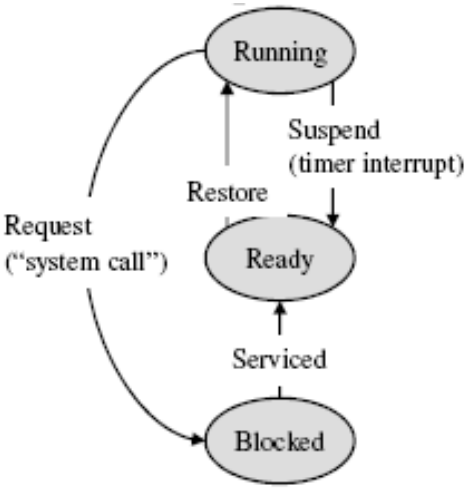
Este último caso sería multiprogramación en un procesador

- 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.***

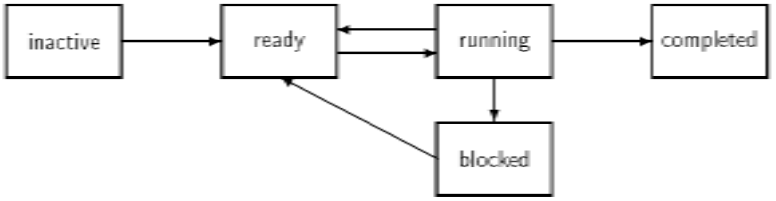
Process switch: suspender el proceso actual y restaurar otro

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.

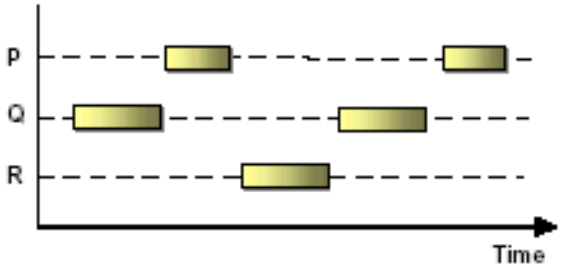


- Estados de los Procesos



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, 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.

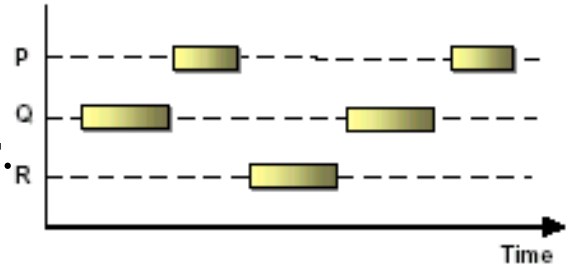
Características importantes:

- interacción
- no determinismo $\Rightarrow$ dificultad para la interpretación y debug

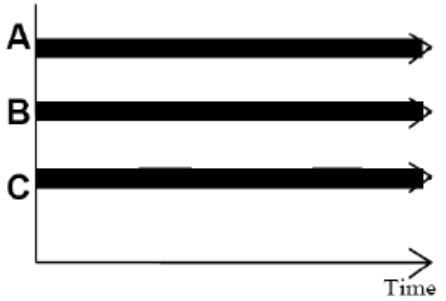
Programa Concurrente: *Concurrencia vs. Paralelismo*

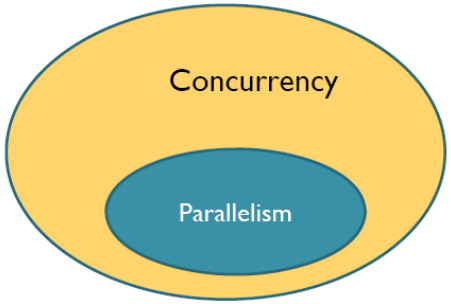
La concurrencia no implica paralelismo

- 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 ⇒** ¿que sucede en este caso?

Ejemplo ⇒
- **x=0; //P**
- **y=0; //Q**
- **z=0; //R**

  - Instrucciones lógicamente concurrentes → Orden de ejecución es irrelevante (Tener la computación distribuida en 3 máquinas sería más rápido).

P → Q → R
Q → P → R
R → Q → P

} Darán el mismo resultado

Secuencialidad y concurrencia

Ejemplo ⇒
- **P ≡ p_1, p_2, p_3, p_4, …**
- **Q ≡ q_1, q_2, q_3, q_4, …**
- **R ≡ r_1, r_2, r_3, r_4, …**

} **P, Q, R independientes.**

- Orden parcial → La única regla es que p_i ejecuta antes que p_j si $i<j$ (idem con R y Q).
- Los siguientes ordenamientos darán el mismo resultado.

p_1,p_2,q_1,r_1,q_2…
q_1,r_1,q_2,p_1…

Ejemplo ⇒ Suponemos que ***x = 5.***
- ***x = 0;*** **//P**
- ***x = x + 1;*** **//Q**

- Orden → Diferentes ejecuciones, pueden dar distintos resultados.

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.

3 grandes clases (superpuestas) de aplicaciones

- Sistemas **multithreaded**.
- Sistemas de **cómputo distribuido**.
- Sistemas de **cómputo paralelo**.

Clases de aplicaciones

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).***

Ejemplos:

- Sistemas de ventanas en PCs o WS
- Sistemas Operativos time-shared y multiprocesador
- Sistemas de tiempo real (por ejemplo, en plantas industriales o medicina)

Clases de aplicaciones

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.

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.

Clases de aplicaciones

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.

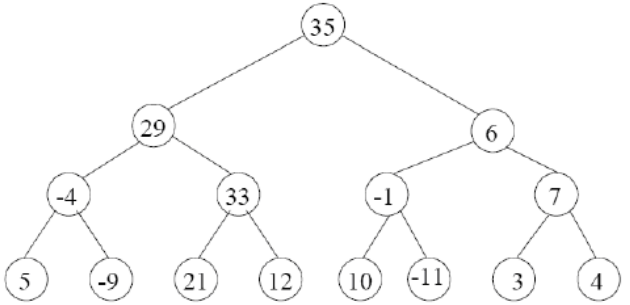
Ejemplos:

- Cálculo científico para modelar y simular sistemas naturales.
- Procesamiento gráfico y de imágenes, efectos especiales, procesamiento de video, realidad virtual.
- Problemas combinatorios y de optimización lineal o no lineal. Modelos econométricos.

Ejemplo simple de concurrencia

Sumar 32 números

- Una persona (*Secuencial)*: 31 sumas, cada una 1 unidad de tiempo.
- Dos personas: cada una suma 16 números. Reducción del tiempo a la mitad. ¿Cuántas unidades de tiempo lleva?
- ¿Que sucede si al aumentar la cantidad de personas?
- Máxima concurrencia: 16 personas
  - Árbol binario
  - ¿Cuántas unidades de tiempo?

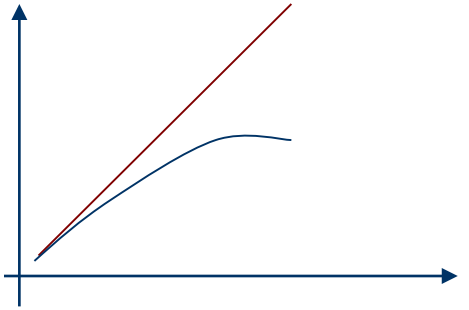


El *cómputo concurrente* con mayor grado de concurrencia resulta en un menor tiempo de ejecución

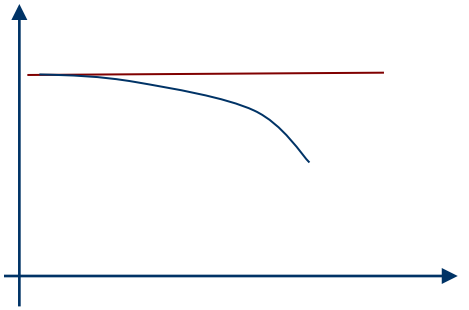
¿Cómo se mide el incremento de performance?

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

- *Speedup* $\Rightarrow S = T_s / T_p$
  - Significado.
  - Rango de valores.



- *Eficiencia* $\Rightarrow E = S / p$
  - Significado.
  - Rango de valores.



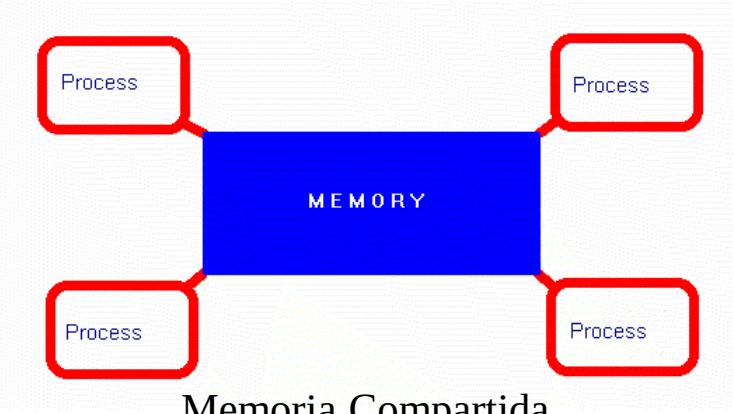
En la ejecución concurrente, el ***speedup*** es menor.

Conceptos básicos de concurrencia

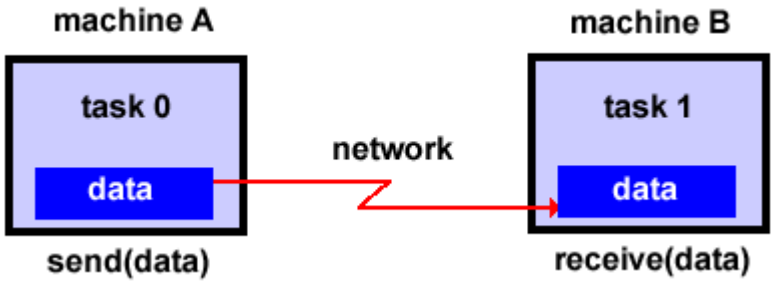
Comunicación entre procesos

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***.
- Por ***Pasaje de Mensajes***.



Memoria Compartida



Pasaje de Mensajes

Conceptos básicos de concurrencia
Comunicación entre procesos

- **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.

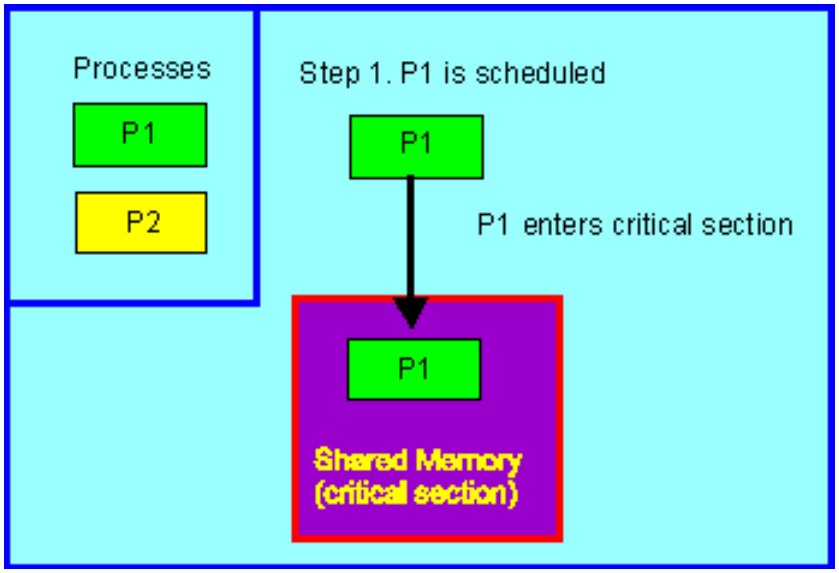
- **Pasaje de mensajes**
  - Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos.
  - También el lenguaje debe proveer un protocolo adecuado.
  - Para que la comunicación sea efectiva los procesos deben “saber” cuándo tienen mensajes para leer y cuando deben trasmitir mensajes.

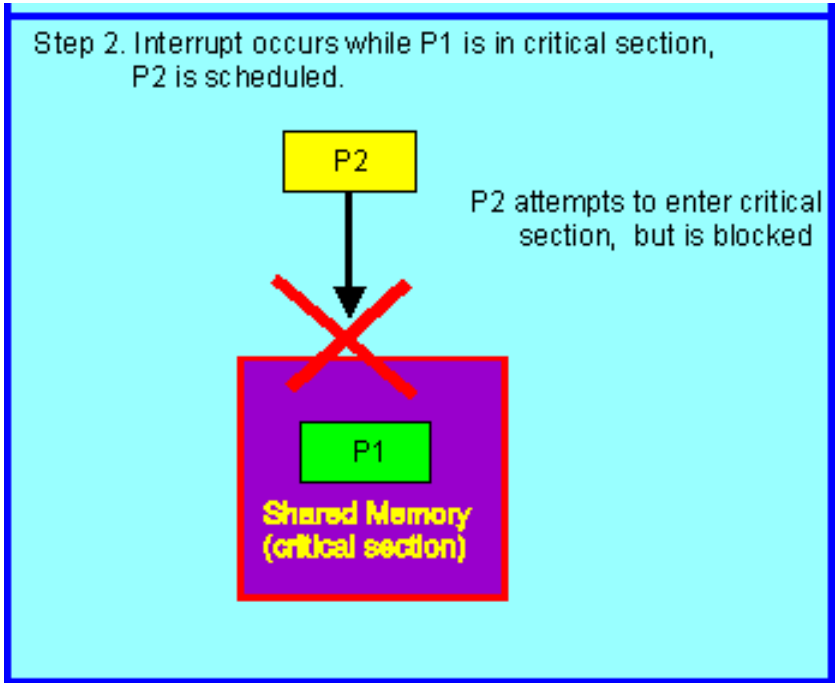
Conceptos básicos de concurrencia

Sincronización entre procesos

La **sincronización** es la posesión de información acerca de otro proceso para coordinar actividades. Los procesos se sincronizan:

- Por *exclusión mutua.*
- Por *condición.*





Ejemplo pasajes de micro.

Conceptos básicos de concurrencia

Sincronización entre procesos

Ejemplo: 3 puertas para entrar a un salón con lugar para *N* personas.

Total = 0;

Puerta - 1
```
{ while (total < N)
      esperar llegada
      total = total + 1;
}
```

Puerta - 2
```
{ while (total < N)
      esperar llegada
      total = total + 1;
}
```

Puerta - 3
```
{ while (total < N)
      esperar llegada
      total = total + 1;
}
```

¿Cuántas personas podrían entrar a la sala ?

Conceptos básicos de concurrencia
Sincronización entre procesos

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.
- Compilar una clase antes de reanudar la ejecución.
- No esperar por algo indefinidamente, si la otra parte ha terminado.

Conceptos básicos de concurrencia
Sincronización entre procesos

- 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.

Conceptos básicos de concurrencia

Sincronización entre procesos

- ***Estado*** de un programa concurrente.
- *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.

Conceptos básicos de concurrencia

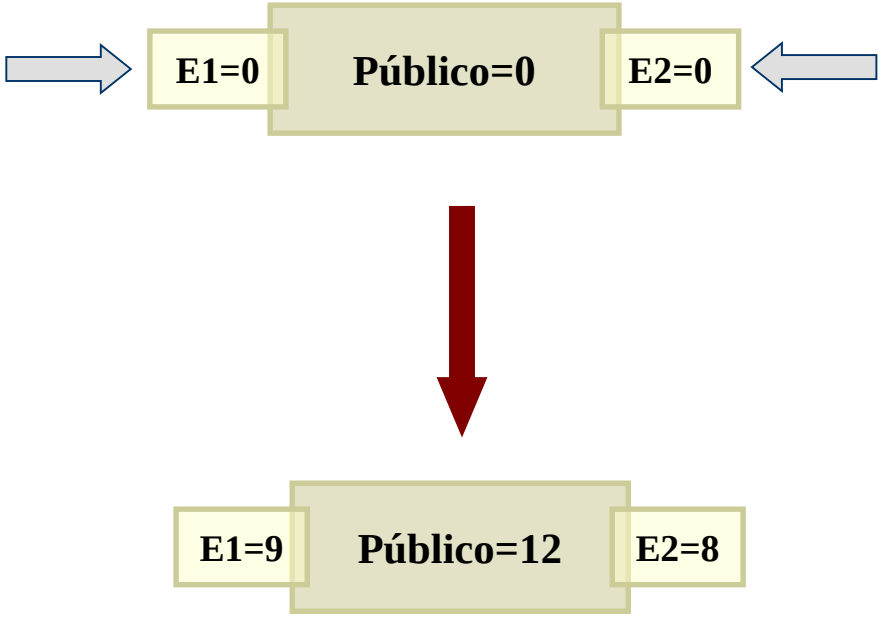
Sincronización entre procesos

***Interferencia*:** un proceso toma una acción que invalida las suposiciones hechas por otro proceso.

Ejemplo: ¿Qué puede suceder con los valores de E1, E2 y público?

```
process 1
  { while (true)
        esperar llegada
        E1 = E1 + 1;
        Público = Público + 1;
  }

process 2
  { while (true)
        esperar llegada
        E2 = E2 + 1;
        Público = Público + 1;
  }
```



Conceptos básicos de concurrencia

Sincronización entre procesos

- Algunas historias son válidas y otras no.

process 1
```
{ while (true)
    p1.1: read(x);
    p1.2: buffer = x;
}
```

process 2
```
{ while (true)
    p2.1: y = buffer;
    p2.2: print(y);
}
```

Posibles historias:

p11, p12, p21, p22, p11, p12, p21, p22, ... ☑
p11, p12, p21, p11, p22, p12, p21, p22, ... ☑
p11, p21, p12, p22, ... ☒
p21, p11, p12, ... ☒

- Se debe asegurar un orden temporal entre las acciones que ejecutan los procesos → las tareas se intercalan ⇒ deben fijarse restricciones.

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

Conceptos básicos de concurrencia

Sincronización entre procesos

La **sincronización** es la posesión de información acerca de otro proceso para coordinar actividades. Los procesos se sincronizan:

- Por *exclusión mutua.*
- 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.

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 básicos de concurrencia

Prioridad y granularidad

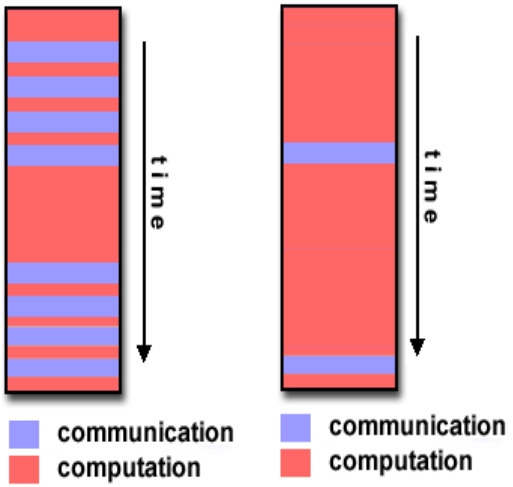
Un proceso que tiene mayor **prioridad** puede causar la suspensión (preemption) de otro proceso concurrente.

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.



Conceptos básicos de concurrencia

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***.

Conceptos básicos de concurrencia

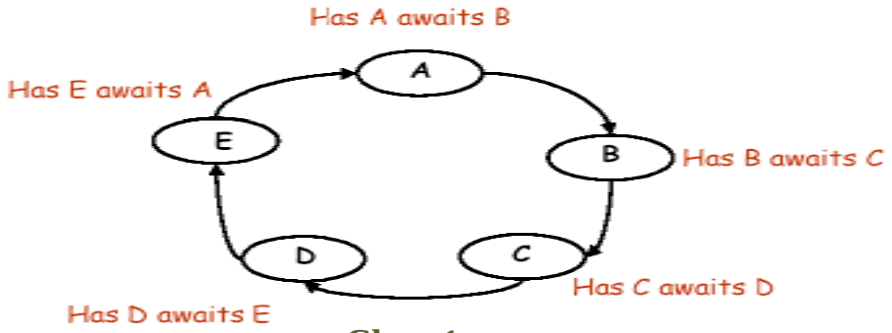
Problema de deadlock

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.

Conceptos básicos de concurrencia
Problema de deadlock

4 propiedades necesarias y suficientes para que exista deadlock son:

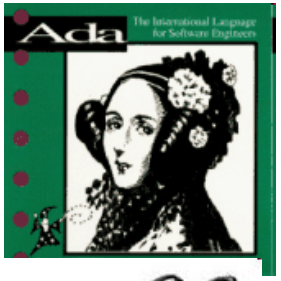
- Recursos reusables serialmente: los procesos comparten recursos que pueden usar con exclusión mutua.
- Adquisición incremental: los procesos mantienen los recursos que poseen mientras esperar adquirir recursos adicionales.
- No-preemption: una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente.
- 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.



Conceptos básicos de concurrencia
Requerimientos para un lenguaje concurrente

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.

- **Requerimientos de un lenguaje de programación concurrente:**
  - 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.
- 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.

Tareas propuestas

- Leer los capítulos 1 y 2 del libro de Andrews
- Leer los ejemplos de paralelismo recursivo, productores y consumidores, clientes y servidores y el código de la multiplicación de matrices distribuida del Capítulo 1 (Andrews).
- Investigar las primitivas de programación concurrente de algún lenguaje de programación.