UNIDAD 3

sábado, 31 de octubre de 2020 7:29

OCTUBRE 30. NOTA: NO HUBO CLASE PERO SE DEBIO VER PARALELISMO.

==========

NOVIEMBRE 6. Por tormenta tropical no hubieron clases.

TEMA: Programación paralela.

****

Nociones de programación paralela.

Creación de un programa paralelo.

- Descomposición.
- Asignación.
- Orquestación.
- Mapeado.

Paralelización de un programa.

****

¿Para quién es importante?

Diseñadores de Algoritmos:

- Diseñar algoritmos que corran bien en sistemas reales.

Programadores:

- Comprender dónde radican las claves del rendimiento para obtener el mejor rendimiento posible en un sistema dado.

Arquitectos:

- Comprender las cargas, las interacciones, y la importancia de los grados de libertad.
- Importante para el diseño y la evaluación.

****

Importancia de la programación.

Diseñamos máquinas para que ejecuten programas. Así pues, éstos:

- ayudan a tomar decisiones en el diseño hardware;
- ayudan a evaluar los cambios en los sistemas.

Son la clave que dirige los avances en la arquitectura uniprocesador.

- Caches y diseño del conjunto de instrucciones.

Mayor importancia en multiprocesadores.

- Nuevos grados de libertad.
- Mayor penalización si hay desacoplo entre programas y arquitectura.
- Más espacio para optimaciones en software.

****

Aun desde el punto de vista arquitectónico, necesitamos abrir la “caja negra” del

software.

Aunque un problema disponga de un buen algoritmo secuencial, paralelizarlo no es trivial.

Hay que ser capaces de:

- extraer el paralelismo y
- combinarlo con las prestaciones ofrecidas por la arquitectura.

****

Ejemplos de problemas paralelizables:

Simulación de corrientes oceánicas.

- Estructura regular de comunicación, computación científica.

Simulación de la evolución de galaxias.

- Estructura irregular de comunicación, computación científica.

Reproducción de escenas por trazado de rayos.

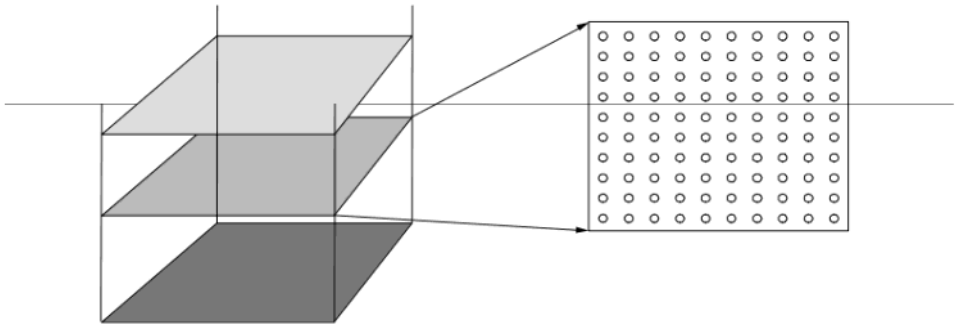
- Estructura irregular de comunicación, computación gráfica.

Data mining.

- Estructura irregular, procesado de información.

****

Simulación de corrientes oceánicas



Modelado como parrillas de dos dimensiones.

Separación en espacio y tiempo.

- Finura en la resolución espacial y temporal => gran precisión.

Muchas operaciones diferentes por unidad de tiempo.

- Planteamiento y solución de ecuaciones.

Concurrencia de computación a través y entre la parrilla.

****

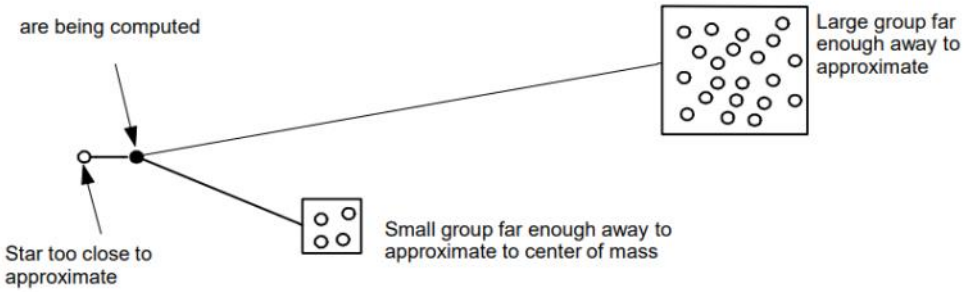
Simulación de la evolución de galaxias

Simula la interacción de multitud de estrellas en el tiempo.

La computación de las fuerzas es cara.

Aproximación bruta: la computación entre pares de estrellas tarda O(n^2).

Los métodos jerárquicos tardan O(n·log n). Se utiliza el hecho de que la fuerza decae con el cuadrado de la distancia.



****

Reproducción de escenas por trazado de rayos.

Dispara rayos dentro de la escena a través de pixels en el plano de la imagen.

Seguimiento de sus caminos:

- Rebotan al chocar con los objetos.
- Generan nuevos rayos: árbol de rayos por rayo entrante.

El resultado es el color y la opacidad de los pixels.

Paralelismo entre rayos.

****

Minería de datos (Data mining)

Extracción de información útil ("conocimiento") a partir de grandes volúmenes de datos.

Un problema particular es el hallazgo de relaciones entre conjuntos en principio no correlacionados de datos.

Existe paralelismo a la hora de examinar conjuntos de datos relacionados de tamaño k — 1 para tratar de obtener un conjunto de datos relacionados de tamaño k.

Video.

https://www.youtube.com/watch?v=QheQowCdWMU

****

Creación de un programa en paralelo

Asumimos: se nos da un algoritmo secuencial.

- Si éste no se presta a la paralelización, se hace necesario un algoritmo completamente diferente.

Proceso:

- Identificar el trabajo que se puede hacer en paralelo.
- Dividir el trabajo y/o los datos entre procesos.

- Nota: El trabajo incluye la computación, el acceso a datos y la entrada/salida.
- Gestionar los accesos a datos, las comunicaciones y la sincronización.

Objetivo:

- Obtener alta productividad con costo bajo en programación y recursos.

****

Objetivo principal: incrementar el Speedup.

$$Speedup\ (p) = \frac{Performance(p)}{Performance(1)}$$

Para un problema fijo:

$$Speedup\ (p) = \frac{Time(1)}{Time(p)}$$

****

Tarea:

- Orden de trabajo que no puede descomponerse en subórdenes ejecutables en paralelo.
- Se ejecuta secuencialmente; la concurrencia solo se da entre tareas.

Proceso (thread o hebra):

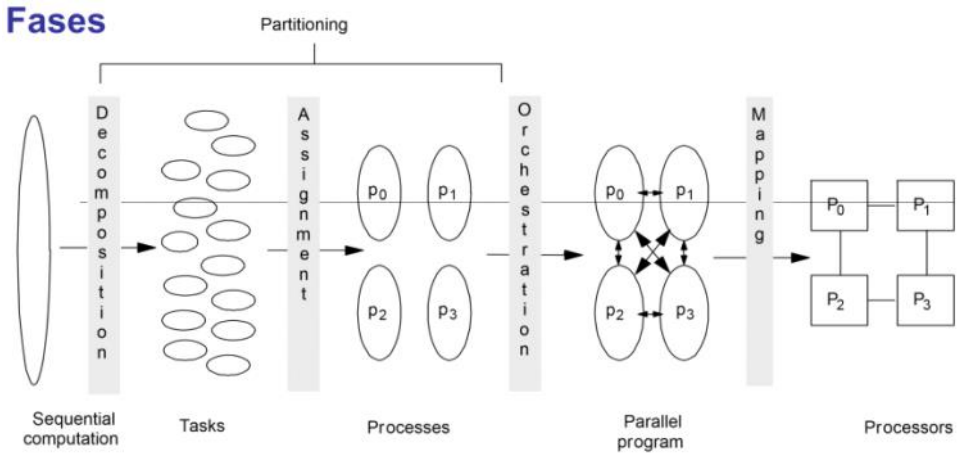
- Entidad abstracta que ejecuta las tareas que le son asignadas.
- Un programa paralelo está constituido por “muchos procesos cooperando”.
- Hay una comunicación y sincronización en los procesos para la ejecución de las tareas.

Procesador:

- Elemento físico en el que se ejecutan los procesos.
- Los procesos constituyen la máquina virtual para el programador.
  - Primero se escribe el programa en términos de procesos y después de mapea a los procesadores.

****

Fases



4 pasos: Descomposición, Asignación, Orquestación, Mapeado.

- Hecho por el programador o por el sistema (compilador, runtime, ...)
- Las acciones son las mismas; si las hace el programador, se realizan de forma explícita.

****

Descomposición

División del programa en tareas que serán distribuidas entre los procesos.

Las tareas pueden estar dispuestas para ejecución dinámicamente.

- El número de tareas disponibles puede variar con el tiempo.
- Identificar la concurrencia y decidir a qué nivel se va a explotar.

Objetivo: Suficientes tareas para mantener ocupados los procesos pero no demasiadas.

- El número de tareas disponibles en un momento define el límite del speedup que podemos conseguir.

****

Límites de Concurrencia: Ley de Amdahl.

Es la limitación fundamental del speedup.

Si la fracción s del programa es inherentemente serie, speedup $\leq 1/s$.

Ejemplo: Supongamos un cálculo en dos fases con p procesadores.

- Primera fase: Operación independiente sobre cada punto de una rejilla de n×n elementos.
- Segunda fase: Suma global de los n^2 puntos.

Tiempo para la primera fase = n ^2 /p.

Tiempo para la segunda fase = n^ 2 .

****

$$Speedup \leq \frac{2n^2}{\frac{n^2}{p} + n^2} = \frac{2p}{p+1}$$

Tips: dividir la segunda fase en dos:

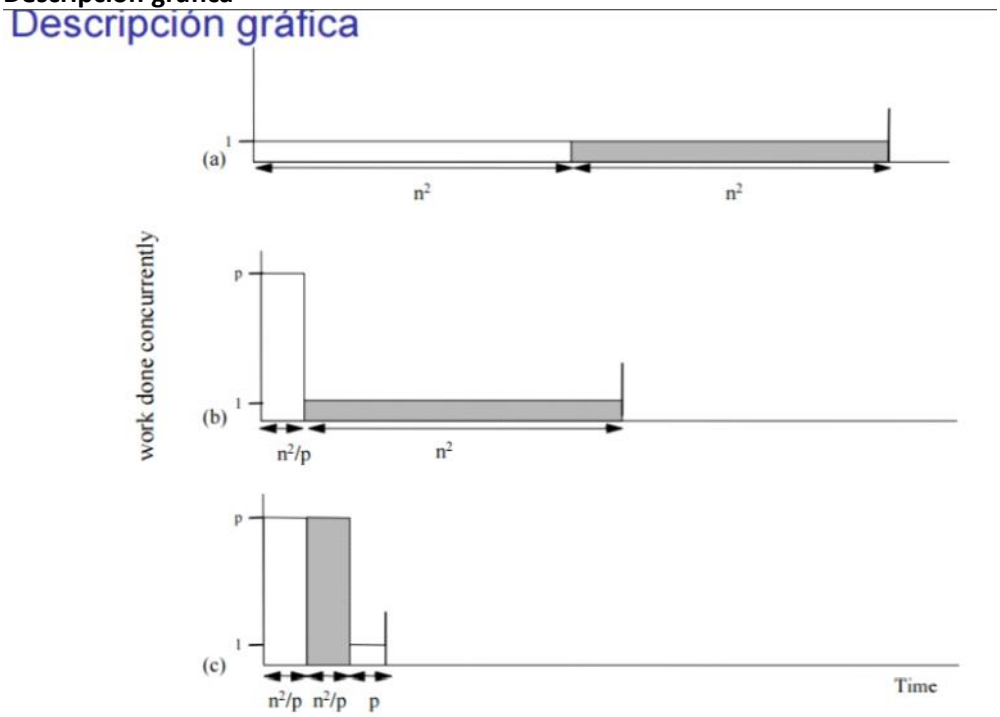
acumular en sumas locales durante el primer barrido,
sumar los valores locales para alcanzar la suma total.

Tiempo $n^2/p + n^2/p + p$, y mejora en *speedup*: $\frac{p\ 2n^2}{2n^2 + p^2}$

si n es muy grande, el speedup es lineal en p.

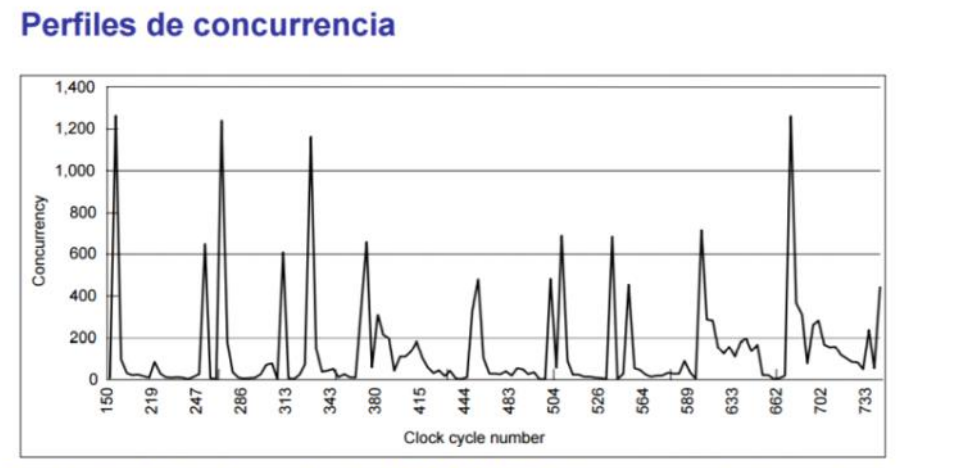
****

Descripción gráfica



****

Perfiles de concurrencia



Simulación lógica de un circuito MIPS R6000. El eje *y* muestra el número de elementos lógicos evaluables para cada ciclo de reloj simulado (eje *x*).

Simulación lógica de un circuito MIPS R6000. El eje y muestra el número de elementos lógicos evaluables para cada ciclo de reloj simulado (eje x).
****

Perfiles de concurrencia

- Área bajo la curva: trabajo total o tiempo para un procesador.
- Extensión horizontal: límite inferior de tiempo (infinitos procesadores).

El speedup es:

$$\text{El } speedup \text{ es: } \frac{\sum_{k=1}^{\infty} f_k k}{\sum_{k=1}^{\infty} f_k \left\lceil \frac{k}{p} \right\rceil}$$

La ley de Amdahl es aplicable a cualquier tipo de sobrecarga y no sólo al límite de concurrencia.
****

Asignación

Mecanismo específico para dividir el trabajo entre procesos.

- Junto con la descomposición recibe el nombre de partición.
- Objetivos:
  - equilibrar la carga;
  - minimizar comunicación y coste de la gestión de la asignación.

Aproximarse a las estructuras suele ser buena idea.

- Inspección del código (paralelizar bucles).
- Existen procesos heurísticos ya establecidos.
- Asignación estática frente a asignación dinámica.

La partición es la primera preocupación del programador.

- Generalmente independiente de la arquitectura o el modelo de programación.
- Pero el costo y complejidad de las primitivas usadas puede afectar a la decisión.

****

Orquestación

Para ejecutar las tareas asignadas, los procesos necesitan mecanismos para:

- referirse y acceder a los datos;
- intercambiar datos con otros procesos (comunicación); sincronizar las actividades.

Las decisiones tomadas en la orquestación son muy dependientes

- del modelo de programación;
- de la eficiencia con que las primitivas del modelo están implementadas.

****
En la orquestación se incluyen, entre otras cosas:

- cómo organizar las estructuras de datos;
- cómo planificar temporalmente las tareas asignadas a un proceso para explotar la localidad de datos;
- cómo organizar la comunicación entre procesos resultante de la asignación.

****

Objetivos

- Reducir el coste de comunicación y sincronización.
- Preservar la referencias a datos locales (organización de las estructuras de datos).
- Programar las tareas para satisfacer rápidamente las dependencias.
- Reducir la sobrecarga que introduce el paralelismo.

Cercanos a la Arquitectura (y al modelo de programación)

- La elección depende mucho de la comunicación, eficiencia y las primitivas.
- La arquitectura debe proporcionar las primitivas adecuadas de manera eficiente.

****
Mapeado

Después de la orquestación ya tenemos un programa paralelo.

Dos aspectos del mapeado:

- Qué procesos se ejecutarán en el mismo procesador.
- Qué procesos se ejecutan en un procesador determinado.
    - El mapeado se realiza en una topología determinada.

****
Un extremo: space-sharing:

- La máquina se divide en subconjuntos con sólo una aplicación en un momento dado en un subconjunto.
- Los procesos pueden ser asignados a un procesador o al sistema operativo.

Otro extremo: el sistema operativo maneja todos los recursos.

El mundo real es una mezcla de los dos.

- El usuario manifiesta deseos en algún aspecto pero el sistema puede ignorarlos.

==========
NOVIEMBRE 13

TEMA: En este día se vio lo de la clase correspondiente a la anterior.

==========

NOVIEMBRE 20

TEMA: Programación paralela

¿Qué es el paralelismo en Programación?

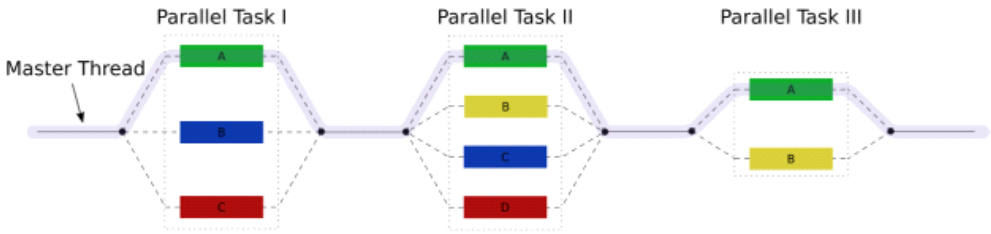
R/
Paralelismo es la ejecución simultánea de dos o más tareas.

¿Se puede considerar una propiedad del hardware, el paralelismo? R/Verdadero.

Se considera una propiedad del hardware, ya que requiere recursos físicos para ejecutar cada tarea simultáneamente,

¿En qué se basa su objetivo?
R/

y su objetivo se basa en realizar una tarea en el menor tiempo posible.



****

Recordando nuevamente…

Concepto de paralelismo:

El paralelismo: acelera la ejecución de una tarea dividiéndola en computaciones independientes y ejecutándola sobre hardware capaz de realizar computaciones simultáneas, como por ejemplo **un procesador con varios núcleos.**

Distribuida. O Paralelismo distribuido.

¿Qué sucede cuando ejecutamos una tarea paralelizada en múltiples ordenadores, en vez de en los múltiples cores de un solo ordenador?
R/
Se dice que la computación paralela es distribuida.

Pero cuando ejecutamos una tarea paralelizada en múltiples ordenadores, en vez de en los múltiples cores de un solo ordenador, decimos que la computación paralela es **distribuida.**

"**Ejemplo**, cada búsqueda en Google **se ejecuta simultáneamente** en cientos de ordenadores, cada uno de los cuales busca al mismo tiempo en un subconjunto del índice del web."
****
¿Qué es la ley de Wirth?

Ley de Wirth

"El Software se expande para ocupar todo el espacio disponible"

Esto quiere decir, que entre más espacio se crea software más pesado o más robustos.

“El software se ralentiza más deprisa de lo que se acelera el hardware.”
****
¿Qué es la Ley de Moore?
Video Ley de Moore.
https://www.youtube.com/watch?v=u3uSuy8ejuI
19 de abril de 1965
El número de transistores en circuitos integrados se duplicaba cada año y que la tendencia seguiría las siguientes dos décadas. Después predijo que cada 2 años durante las siguientes dos décadas y tuvo razón de nuevo.

Cofundador de Intel y Fairchild Semiconductor.

Fin de la ley de moore se cree que se verá en 2025, con los transistores de un nanómetro.
¿Por qué se desea crear chips con transistores más pequeños?
Entre más transistores se tenga mayor es el poder computacional, entre menor sean estos transistores menor es el voltaje que requieren para funcionar y el tiempo de carga y descarga mejora y serían más rápidos al momento de procesar algo. Sin embargo al haber más transistores el calor será un problema, a pesar del menor voltaje utilizado.

El calor generado por un cpu o gpu es un problema actual.

Se requiere de una nueva tecnología para dar el siguiente paso con los circuitos integrados como:
Procesadores cuánticos.
Procesadores neuromórficos.
Mejorar algoritmos para sacarle provecho a la computación en paralelo y darle un mejor futuro a los circuitos integrados actuales y evitaríamos tener que hacerlos tan pequeños

****

Ley de Wirth

El hardware se está volviendo, claramente, más rápido a medida que pasa el tiempo y parte de ese desarrollo está cuantificado por la **Ley de Moore.**

Los programas tienden a hacerse más grandes y complicados con el paso del tiempo y a veces los programadores se refieren a la Ley de Moore para justificar la escritura de código lento o no optimizado,

pensando que no será un problema porque el hardware sobre el que correrá el programa será cada vez más rápido.

****
Ejemplo de la ley de Wirth

Un ejemplo de la Ley de Wirth que se puede observar es:

◦ Que el tiempo que le toma a un PC actual **arrancar su sistema operativo** no es menor al que le tomaría a un PC de hace cinco o diez años con un sistema operativo de la época.

"Porque los programadores de aplicaciones, en la mayoría de los casos, no sabemos emplear la potencia del hardware y no se optimiza"

****

Multitarea

Hacer varias cosas de manera simultánea, que gran regalo y que lio

No, solo hay que entender en que se basa y como se programa.

****

¿Para qué queremos la multitarea?

Una respuesta rápida sería:

◦ Para que el usuario no se enfade con nosotros, por esperar demasiado, en momentos en los que no hace falta

◦ Notar que nos referimos la mayoría de las veces **del bloqueo al usuario,** no a la ejecución que se produce debajo de esa pantalla de carga.

****

Otro efecto molesto es aquel que se da cuando el usuario pulsa un botón y la aplicación se bloquea, pero no se bloquea por un error, sino porque debajo se está ejecutando un montón de código que requiere tiempo de proceso para concluirse.

Todo esto se soluciona con la multitarea, pero puede que antes haya que pensar un poco para en la solución con ésta.

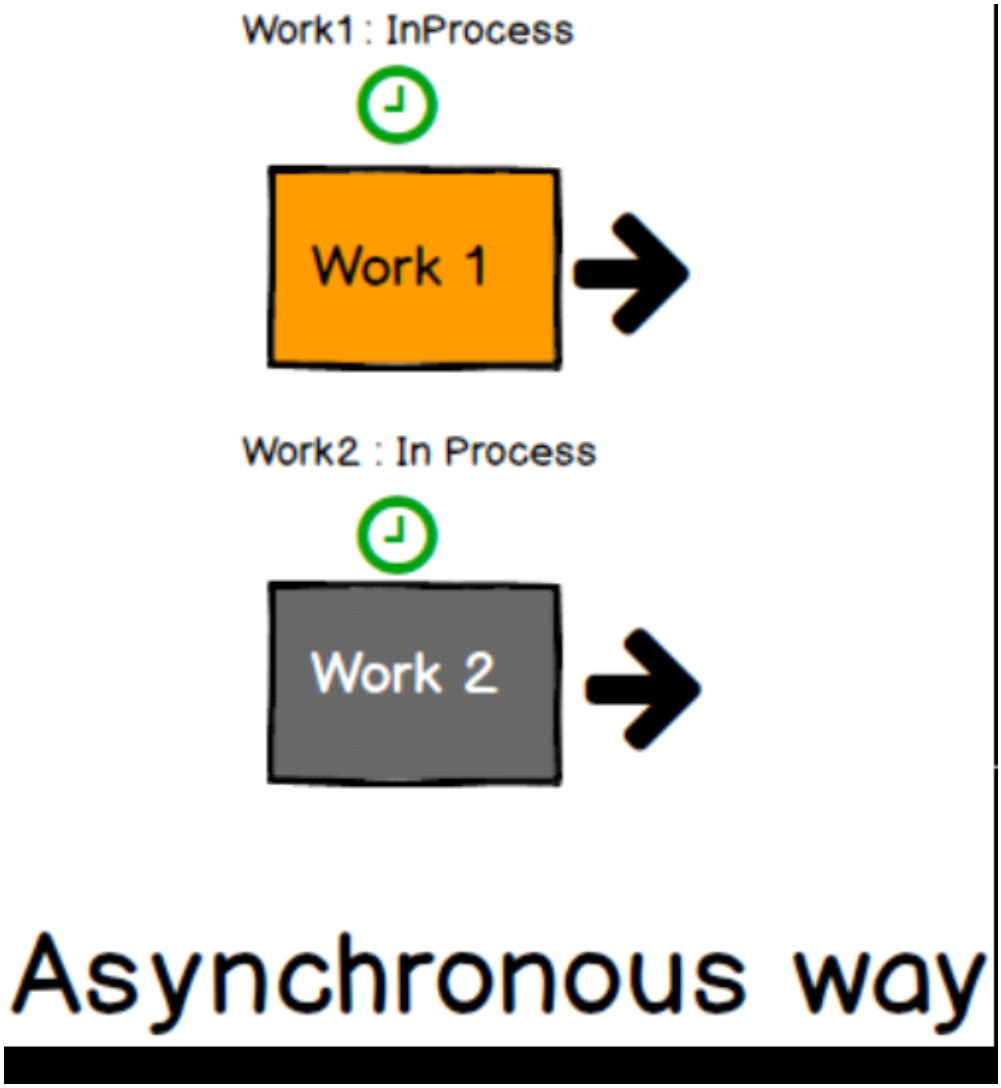
****

Con la multitarea el usuario nunca quedará bloqueado –Logrando de esta manera seguir usando la aplicación mientras algo muy gordo se ejecuta debajo;

se eliminan la mayoría de pantallas de carga o no interrumpirán la experiencia de uso de la aplicación; y se ejecutará de manera más óptima, haciendo que el procesador no esté esperando continuamente y con cuellos de botella por llegarle un montón de cosas a la vez.

****

Threads



Hilos

¿Qué es un hilo? R/

Un Hilo es un trozo de código de nuestro programa que puede ser ejecutado al mismo tiempo que otro.

¿Qué puede ejecutarse de manera simultánea a otro?

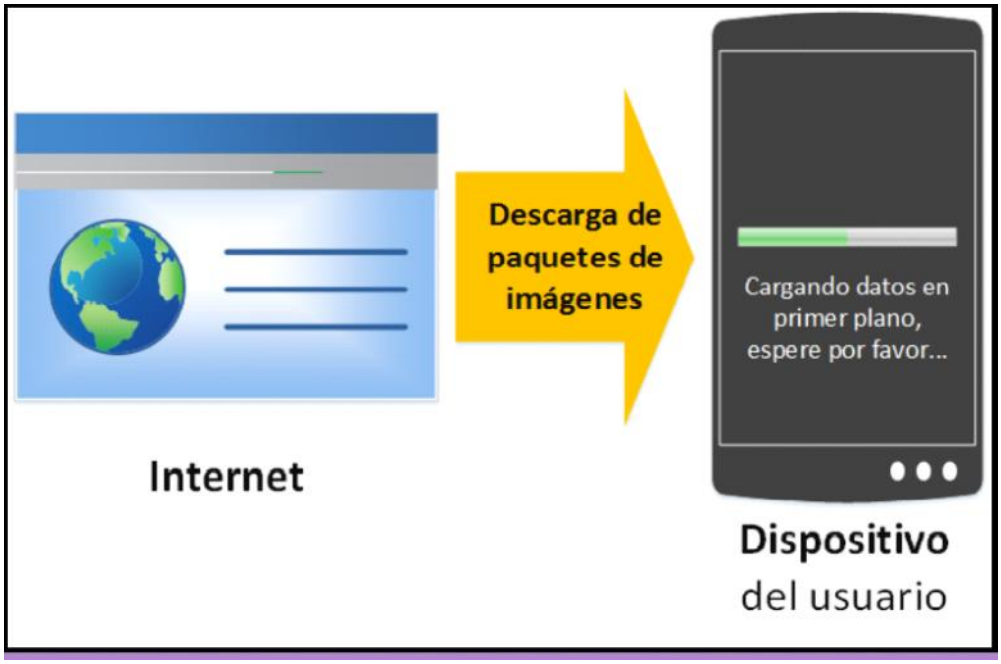
por ejemplo, queremos ver un listado de 100 imágenes que se descargan desde Internet,

como usuario

¿Cuál de las dos opciones siguientes elegirías?

[Escenario 1]

Descargar las imágenes 100 imágenes, haciendo esperar al usuario con una pantalla de “cargando” hasta que se descargan todas. Luego podrá ver el listado con las imágenes.



[Escenario 2]

Que mientras se descargan las 100 imágenes, el usuario pueda ir viendo y usando las que ya se han descargado.

****

Hilos

Como desarrollador la opción A

“Es más sencilla que un usuario preferiría. “

La B es lo que todo usuario quiere de una aplicación:

“Tener que esperar no es una opción. “

Volviendo al punto de vista del desarrollador, **tampoco es una opción**.

Un buen desarrollador de aplicaciones hace bien las cosas y se inclina por la opción B.

la opción A no existe, la opción A nos lleva de cabeza a la “ley de Wirth” antes descrita- queremos ser profesionales y la opción B es la opción a aplicar.

****

Dos áreas bien diferenciadas desde el punto de vista del usuario.

Desde el punto de vista del usuario existen **dos áreas bien diferenciadas**, que el desarrollador ha de tener en cuenta:

◦ **Primer plano (se ejecuta únicamente un hilo):** Aquí se ejecuta únicamente un hilo llamado “hilo principal”.

◦ **Segundo plano** (o en inglés **background**): Se ejecuta todo el resto de hilos.
En el background o segundo plano se ejecutan el resto de hilos.

****

Ejemplo

Hilos - Thread.

El hilo viene siendo un proceso que se ejecuta.

Todo programa, por ejemplo en C# hay un hilo principal y se puede identificar por el **static void Main** . Este es un hilo y es una ejecución de un programa.

Los hilos pueden trabajar como subprocesos.

Un hilo es un subproceso que se ejecuta dentro de un proceso principal.

Los hilos sirven para repartir el trabajo que se ejecute en paralelo y ahorrar tiempo

Del label al textbox de dos formas.

```
11  namespace wfa
12  {
13      public partial class Form1 : Form
14      {
15          public Form1()
16          {
17              InitializeComponent();
18              CheckForIllegalCrossThreadCalls = false;
19          }
20          public void go()
21          {
22              textBox1.Text = label1.Text;
23          }
24
25          private void button1_Click(object sender, EventArgs e)
26          {
27              Thread t = new Thread(go);
28              t.Start();
29          }
30      }
31  }
```

No es la más recomendable porque se traba.

2.Forma segura

```
namespace wfa
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public void go()
        {
            if (InvokeRequired)
                Invoke(new Action(() => textBox1.Text = label1.Text));
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Thread t = new Thread(go);
            t.Start();
        }
    }
}
```

Fin.

Video.
https://www.youtube.com/watch?v=QgvzDdFpARM

La programación paralela no tiene sentido si no es para acelerar una aplicación.

Programación Paralela.

-(Tradicionalmente) uso de **varios computadores** trabajando juntos para resolver una **tarea común:**

-Cada computador trabaja en una **porción del problema.**

-Los procesos pueden **intercambiar datos y comunicarse**, a través de:

-Memoria (**Modelo de memoria compartida**)

-Red de interconexión (**Modelo de Paso de mensajes**)

-(Actualmente) uso de **varios procesadores** trabajando juntos para resolver una **tarea común.**

-Estos procesadores pueden estar **dentro del mismo ordenador y tener**

distintas naturalezas (heterogéneos).

-E incluso del mismo chip (multicore, GPUs, etc)

En la programación paralela ya se puede desarrollar dentro de un mismo chip. Ya no es necesario comunicar dos computadores, sino que dentro del mismo ordenador se puede tener cientos de procesadores.

Algunas definiciones previas

Proceso: Programa en ejecución. Desde el punto de vista del procesador **conjunto de instrucciones** a ejecutar sobre unos datos de entrada.

Hilo de ejecución: Un **proceso está compuesto por uno o más hilos de ejecución.** Los hilos de un proceso se **dividen** el trabajo a realizar por el proceso y pueden cooperar para solucionar el problema.

-El **programador (compilador)** es responsable de crear los hilos y asignarles trabajo.

==========

NOVIEMBRE 27

TEMA

****

PARA EL EXAMEN:

Va a poner un fragmento de código y que le digamos como sería el orden o sea el resultado.