2ºDAM - Programación de servicios y procesos

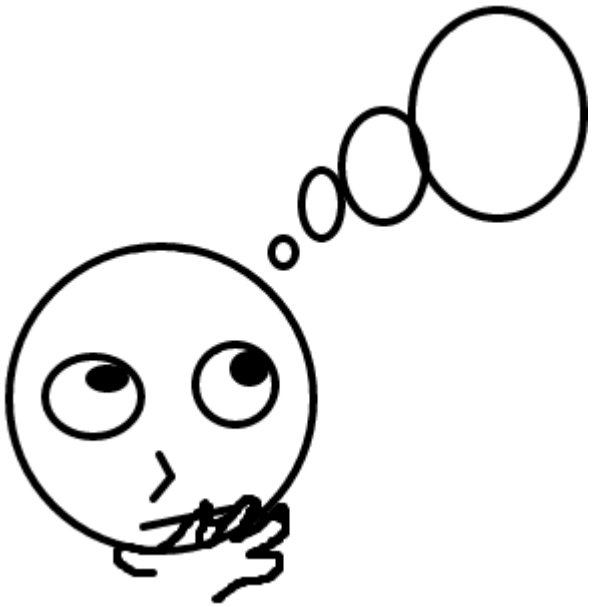
Tema 2 – Programación multiproceso

CONTENIDOS

1. Programación concurrente, paralela y distribuida
2. Elementos de un proceso.
3. La necesidad de comunicación entre procesos
4. Mecanismos de comunicación y/o sincronización entre procesos
   - Señales
   - Tuberías
   - Semáforos
   - Colas de mensajes
   - Segmentos de memoria compartida

1. Programación concurrente, paralela y distribuida

1. ¿Es lo mismo programación concurrente que paralela?
2. ¿Es lo mismo programación concurrente que distribuida?
3. ¿Es lo mismo programación paralela que distribuida?



1. 1. Programación concurrente

Programación concurrente.

- De una manera aproximada podemos decir que la concurrencia en informática es "***La existencia simultánea de varios procesos en ejecución***".

- **Dos procesos serán concurrentes cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última.** Es decir, existe un solapamiento o intercalado en la ejecución de sus instrucciones.

1. 1. Programación concurrente

- Por otro lado, hemos visto que los procesos activos se ejecutan alternando sus instantes de ejecución en la CPU. Y, aunque nuestro equipo tenga más de un núcleo, los tiempos de ejecución de cada núcleo se repartirán entre los distintos procesos en ejecución. **La planificación alternando los instantes de ejecución en la gestión de los procesos, hace que los procesos se ejecuten de forma concurrente.** O lo que es lo mismo: *multiproceso* **= *concurrencia***.

1. 1. Programación concurrente

- La **programación concurrente** proporciona **mecanismos de comunicación** y **sincronización** entre procesos que se ejecutan de forma simultanea en un sistema informático.
- La programación concurrente nos permitirá definir qué **instrucciones** de nuestros procesos se pueden ejecutar de forma simultánea con las de otros procesos, sin que se produzcan errores; y cuáles deben ser **sincronizadas** con las de otros procesos **para que los resultados de sean correctos**.

1.1. Programación concurrente

Sabías que…

¿Sabías que los sistemas operativos de Microsoft no fueron multiproceso hasta la aparición de Windows 95 (en 1995)?

- **MS-DOS** era un **sistema operativo monousuario y monotarea**. Los programadores, no el sistema operativo, implementaban la alternancia en la ejecución de las distintas instrucciones de los procesos que constituían su aplicación para conseguir interactuar con el usuario. Lo conseguían capturando las interrupciones hardware del equipo.
- Además, MS-DOS, no impedía que unos procesos pudieran acceder al espacio de trabajo de otros procesos, e incluso al espacio de trabajo del propio sistema operativo.

1.1. Programación concurrente

- Por otro lado UNIX, que se puede considerar 'antepasado' de los sistemas GNU/Linux, fue diseñado **portable, multitarea, multiusuario** y en **red** desde su origen en 1969.

1. 1. Programación concurrente

Concurrencia y hardware

En un sistema **monoprocesador** se puede tener una ejecución concurrente gestionando el tiempo de procesador para cada proceso. El S.O. va alternando el tiempo entre los distintos procesos, cuando uno necesita realizar una operación de entrada salida, lo abandona y otro lo ocupa; de esta forma se aprovechan los ciclos del procesador. Esta forma de gestionar los procesos en un sistema monoprocesador recibe el nombre de **multiprogramación**.

1. 1. Programación concurrente

Concurrencia y hardware

En un sistema **monoprocesador** todo los procesos comparten la misma memoria. La forma de comunicar y sincronizar procesos se realiza mediante ***variables compartidas***.

En un sistema **multiprocesador** podemos tener un proceso en cada procesador. Esto permite que exista **paralelismo** real entre los procesos.

1.1. Programación concurrente

Beneficios de la programación concurrente:

- **Mejor aprovechamiento de la CPU**. Un proceso puede aprovechar ciclos de CPU mientras otro realiza una operación de entrada / salida.
- **Velocidad de ejecución**. Al subdividir un programa en procesos, estos se pueden "repartir" entre procesadores o gestionar en un único procesador según importancia.

1.1. Programación concurrente

Beneficios de la programación concurrente:

- **Solución a problemas de naturaleza concurrente**. Existen algunos problemas cuya solución es más fácil utilizando esta metodología:
  - **Sistemas de control.** Son sistemas en los que hay captura de datos, normalmente a través de sensores, análisis y actuación en función del análisis. Un ejemplo son los sistemas en tiempo real
  - **Tecnologías web.** Los servidores web son capaces de atender múltiples peticiones de usuarios concurrentemente, también los servidores de chat, correo, los propios navegadores web, etc.

1.1. Programación concurrente

Beneficios de la programación concurrente:

- **Más soluciones a problemas de naturaleza concurrente**.
  - **Aplicaciones basadas en GUI.** El usuario puede interactuar con la aplicación mientras la aplicación está realizando otra tarea. Por ejemplo el navegador web puede estar descargando un archivo mientras el usuario navega por las páginas.
  - **Simulación.** Programas que modelan sistemas físicos con autonomía.
  - **Sistemas gestores de bases de datos.** Los usuarios interactúan con el sistema, cada usuario puede ser visto como un proceso.

1.1. Programación concurrente

Problemas inherentes a la programación concurrente

Exclusión mutua. En programación concurrente es muy típico que varios procesos accedan a la vez a una variable compartida para actualizarla. Esto se debe evitar, ya que puede producir inconsistencia de datos.

Uno puede estar actualizando la variable a la vez que otro la puede estar leyendo. Por ello ***es necesario conseguir la exclusión mutua de los procesos respecto a la variable compartida***.

1.1. Programación concurrente

Problemas inherentes a la programación concurrente

Para ello se propuso la **región crítica**. Cuando dos o más procesos comparten una variable, el acceso a dicha variable debe efectuarse siempre dentro de una región crítica asociada a la variable. Solo uno de los procesos podrá acceder para actualizarla y los demás deberán esperar. El tiempo de estancia es finito.

1.1. Programación concurrente

Problemas inherentes a la programación concurrente

Ejemplo:

```
int var = 7;
while(var <= 100){
    printf("%i", var);
    var++;
}
```

Si mientras sale ese proceso entra otro y aumentar *var* en 71 (por ejemplo), el primer proceso ya no se llegará a ejecutar 93 veces, porque la variable ha sido alterada por otro proceso. Este trozo de código es la región crítica.

1. 1. Programación concurrente

Problemas inherentes a la programación concurrente

Condición de sincronización. Hace referencia a la necesidad de coordinar los procesos con el fin de sincronizar sus actividades. Puede ocurrir que un proceso P1 llegue a un estado X que no pueda continuar su ejecución hasta que otro proceso P2 haya llegado a un estado Y de su ejecución. La programación concurrente proporciona mecanismos para bloquear procesos a la espera de que ocurra un evento y para desbloquearlos cuando este ocurra.

1. 1. Programación concurrente

Problemas inherentes a la programación concurrente

Condición de sincronización.

Algunas herramientas para manejar la concurrencia son:

- La región crítica
- Los semáforos
- Monitores
- etc

Que veremos más adelante en el curso.

1.2. Programación paralela

El **procesamiento paralelo** permite que muchos elementos de procesos independientes trabajen simultáneamente para resolver un problema. Estos elementos pueden ser un número arbitrario de equipos conectados por una red, un único equipo con varios procesadores o una combinación de ambos.

1.2. Programación paralela

En un sistema **multiprocesador** podemos tener un proceso en cada procesador y todos juntos trabajan para resolver un problema. Cada procesador realiza una parte del problema y necesita intercambiar información con el resto. **Según cómo se realice el intercambio** tenemos distintos **modelos de programación paralela**:

- **Modelo de memoria compartida**: los procesadores comparten físicamente la memoria.

1.2. Programación paralela

- **Modelo de paso de mensajes**: cada procesador dispone de su propia memoria independiente al resto y accesible sólo por él. Para realizar el intercambio de información es necesario que cada procesador realice la petición de datos al procesador que los tiene y éste haga el envío.

1. 2. Programación paralela

Ventajas

- Proporciona ejecución simultánea de tareas.
- Disminuye el tiempo total de ejecución de una aplicación.
- Resolución de problemas complejos y de grandes dimensiones.
- Utilización de recursos no locales, por ejemplo los recursos que están en una red distribuida, una WAN o la propia red Internet.
- Disminución de costes, en vez de gastar en un supercomputador muy caro se pueden utilizar otros recursos más baratos disponibles remotamente.

1. 2. Programación paralela

Inconvenientes

- Los compiladores y entornos de programación para sistemas paralelos son más difíciles de desarrollar.
- Los programas paralelos son más difíciles de escribir.
- El consumo de energía de los elementos que forman el sistema.
- Mayor complejidad en el acceso a los datos.
- La comunicación y la sincronización entre diferentes subtareas.

1. 2. Programación paralela

Ejercicio

Entra en la URL:
https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

Responde a las siguientes cuestiones:

- Cita algunas características de la computación serie.
- Cita algunas características de la computación en paralelo.
- Ámbitos en los que se usa la computación en paralelo.

I. 3. Programación distribuida

Uno de los motivos principales para construir un sistema distribuido es compartir recursos. El más conocidos por todos es Internet que permite a los usuarios donde quiera que estén hacer uso de la World Wide Web, correo electrónico y transferencia de archivos.

Un sistema distribuido es aquel en el que los componentes hardware o software, localizaos en computadores unidos mediante una red, comunican y coordinan sus acciones mediante el paso de mensajes.

1. 3. Programación distribuida

Esta definición de sistema distribuido tiene las siguientes **consecuencias**:

- **Concurrencia**: lo usual en una red de ordenadores es la ejecución de programas concurrentes.
- **Inexistencia de reloj global**: no hay una temporalización, los relojes de los host no están sincronizados y los programas cooperan coordinando sus acciones mediante el paso de mensajes.
- **Fallos independientes**: cada componente del sistema puede fallar independientemente, permitiendo que los demás continúen su ejecución.

I. 3. Programación distribuida

El paradigma de la programación distribuida es el resultado natural del uso de las computadoras y las redes.

Modelos de programación para la comunicación entre los procesos de un sistema distribuido:

- **Sockets**. Proporcionan los puntos externos para la comunicación entre procesos. Es actualmente la base de la comunicación. Pero al ser de muy bajo nivel de abstracción, son adecuados a nivel de aplicación.

1. 3. Programación distribuida

- **Llamada de procedimientos remotos o RPC** (Remote Procedure Call). Permite a un programa cliente llamar a un procedimiento de otro programa en ejecución en un proceso servidor. El proceso servidor define en su interfaz de servicio los procedimientos disponibles para ser llamados remotamente.

1. 3. Programación distribuida

- **Invocación remota de objetos**. El modelo de programación basado en objetos ha sido extendido para permitir que los objetos de diferentes procesos se comuniquen uno con otro por medio de una invocación a un método remoto o RMI (Remote Method Invocation). Un objeto que vive en un proceso puede invocar métodos de un objeto que reside en otros procesos. Java RMI extiende el modelo de objetos de Java para proporcionar soporte de objetos distribuidos en lenguaje Java.

1. 3. Programación distribuida

Ventajas

- Se puede compartir recursos y datos
- Capacidad de crecimiento incremental
- Mayor flexibilidad al poderse distribuir la carga de trabajo entre diferentes ordenadores.
- Alta disponibilidad
- Soporte de aplicaciones inherentemente distribuidas
- Carácter abierto y heterogéneo.

1. 3. Programación distribuida

Inconvenientes

- Aumento de la **complejidad**, se necesita un nuevo tipo de software.
- Problemas con las **redes** de comunicación: pérdida de mensajes, saturación del tráfico.
- Problemas de **seguridad** como por ejemplo ataques de denegación de servicio en la que se bombardea un servicio con peticiones inútiles de forma que un usuario interesado en usarlo no pueda emplearlo.

1. 4. Programación concurrente, paralela y distribuida

- **Resumen programación concurrente, paralela y distribuida**

Programación concurrente: tenemos varios elementos de procesos (hilos, procesos) que trabajan de forma conjunta en la resolución de un problema. Se suele llevar a cabo en un único procesador o núcleo.

Programación paralela: es programación concurrente cuando se utiliza para acelerar la resolución de los problemas, normalmente usando varios procesadores o núcleos.

Programación distribuida: es programación paralela cuando los sistemas están distribuidos a través de una red (una red de procesadores); se usa paso de mensajes.