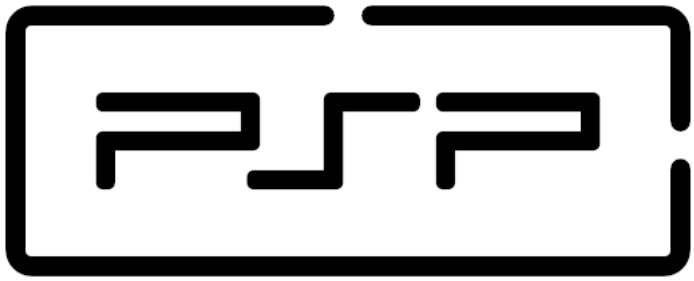


Programación de Servicios y Procesos

1.2. Concurrencia

1.2. Concurrencia

Según el diccionario de la RAE (https://dle.rae.es/concurrencia?m=form) una de las acepciones de concurrencia es

> Coincidencia, concurso simultáneo de varias circunstancias.

Si cambiamos circunstancias por `procesos`, ya tendríamos una definición cercana a lo que significa concurrencia en el mundo digital

Si nos fijamos, no es la primera vez que surge la palabra `proceso` en este texto, y es que los procesos son una pieza fundamental del puzle, por no decir la parte más importante.

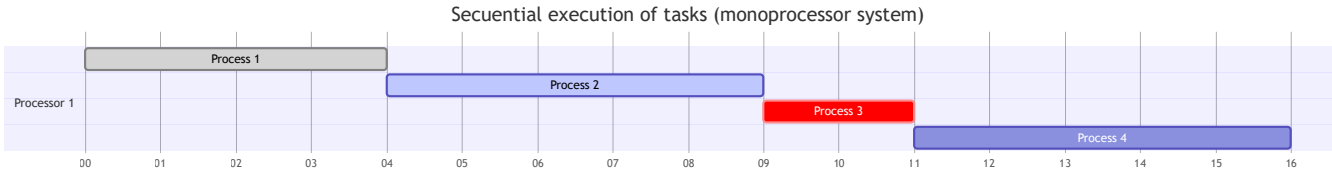
1.2.1. Concurrencia vs Paralelismo

Ahora que ya sabemos qué es un proceso, vamos a ver la relación que éstos tienen con el hardware en el que se ejecutan.

Monoproceso

Por mucho que tengamos varios procesos procesos ejecutándose a la vez, si sólo tenemos un microprocesador para atenderlos a todos, estas tareas nunca van a poder ejecutarse a la vez.

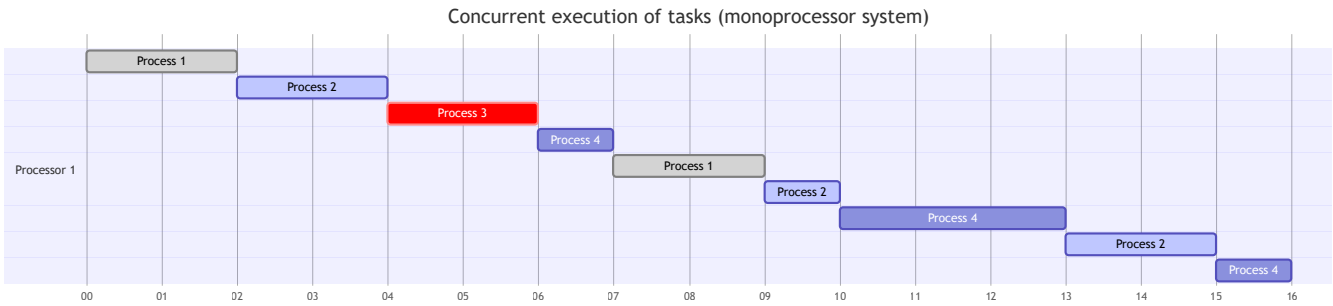
Una posibilidad sería la ejecución secuencias de las tareas en el sistema. Se empieza a ejecutar una tarea y, hasta que esta no finaliza, el sistema no empieza a ejecutar la siguiente. Esto se correspondería con sistemas que sólo son capaces de hacer una tarea a la vez, algo raro hoy en día.



Multiprogramación

Para que los procesos no tengan que esperar a que todos los demás se ejecuten, los sistemas aprovechan y exprimen los recursos al máximo, permitiendo la ilusión de que varios procesos se ejecutan de forma simultánea. Esto es lo que se conoce como `multitarea`.

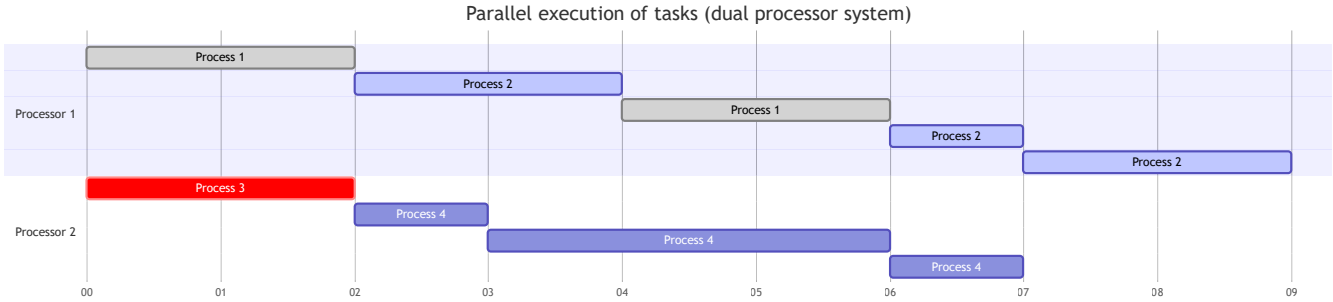
En estos sistemas, se aprovecha el diseño de los sistemas operativos modernos, y de las operaciones que realizan los procesos que no requieren el uso del procesador (esperar a una operación de E/S, una interacción con el usuario, la recepción de información desde la red, etc.) para poder ejecutar otros procesos. La ejecución se multiplexa en el tiempo.



Como se puede observar en las dos imágenes anteriores (aunque se trata sólo de un modelo), el tiempo de uso total del procesador es igual en ambos casos, es decir, que el sistema tardará el mismo tiempo en completar todas las tareas. Sin embargo, la sensación es que todas las tareas se están realizando a la vez.

Paralelismo

Con el avance de la tecnología ahora la gran mayoría de dispositivos, desde los equipos de escritorio, portátiles, dispositivos móviles, ... hasta los dispositivos IoT, tienen capacidades de multiproceso, es decir, tienen más de un procesador para poder realizar varias tareas a la vez de forma real, no simulada. A este tipo de ejecución es a lo que llamamos `paralelismo`.



En este caso, a mayor número de unidades de proceso, menor tiempo tardarán las tareas en completarse y mayor será la sensación de rapidez que notará el usuario. Este es uno de los retos de los sistemas operativos, planificar adecuadamente las tareas para minimizar los tiempos de ejecución, de espera y el uso de los recursos del sistema, el procesador principalmente.

núcleos vs hilos

Si habéis comprado un procesador hace poco, o estáis al día en cuanto al hardware, sabréis que una de las características de los procesadores es su **número de núcleos** (4, 8, 16).

Pero además, al número de núcleos lo acompaña otra característica que es el número de **hilos o threads**, que suele ser el doble que el de núcleos.

¿Qué implicación tienen los threads de un procesador con respecto a la concurrencia? ¿Si un equipo tiene 8 núcleos / 16 hilos significa eso que puede ejecutar 16 procesos a la vez?

1.2.2. Sistemas distribuidos

> "Un sistema distribuido es una colección de computadores independientes que aparecen ante los usuarios como un único sistema coherente"
> "Andrew S. Tanembaum"

Posiblemente el ejemplo más famoso y conocido de sistema distribuido sea `Internet`.Internet aparece ante los usuarios como un enorme repositorio de documentos, es decir, como un único sistema capaz de proveer casi cualquier tipo de información o servicio que se necesite. No obstante, sabemos que está compuesta por millones de equipos ubicados en localizaciones diferentes e interconectados entre sí.

Nace de la necesidad de compartir recursos. Actualmente el máximo exponente de este tipo de sistemas es el `Cloud Computing` o servicios en la nube. Un sistema es distribuido cuando los componentes software están distribuidos en la red, se comunican y coordinan mediante el paso de mensajes.

Las características de este tipo de sistemas son::

- Concurrencia: ejecución de programas concurrentes.
- Inexistencia de un reloj global. Implica sincronizarse con el paso de mensajes.
- Fallos independientes: cada componente del sistema puede fallar sin que perjudique la ejecución de los demás.

1.2.3. Ventajas e inconvenientes

Ventajas del procesamiento paralelo:

- Ejecución simultánea de tareas.
- Disminuye el tiempo total de ejecución
- Resuelve problemas complejos y de grandes dimensiones.
- Utilización de recursos no locales distribuidos en la red
- Disminución de costos, aprevechando los recursos distribuidos, no es necesario gastar en un único supercomputardor, se puede alcanzar el mismo poder de computación con equipos más modestos distribuidos.

Inconvenientes del procesamiento paralelo:

- Los compiladores y entornos de programación para sistemas paralelos son más complicados de desarrollar.
- Los programas paralelos son más difíciles de escribir
- Hay mayor consumo de energía
- Mayor complejidad en el acceso a datos
- Complejidad a la hora de la comunicación y sincronización de las diferentes subtareas. cuidado

Ventajas de la programación distribuida

- Se comparten recursos y datos
- Crecimiento bajo demanda
- Mayor flexibilidad para distribuir la carga
- Alta disponibilidad
- Soporte de aplicaciones distribuidas
- Filosofía abierta y hetereogénea

> **Escalado de sistemas**
>
> Con escalado nos referimos a la posibilidad de incrementar las capacidades de un sistema.
>
> Investiga las diferencias, ventajas e inconvenientes del `escalado horizontal` y el `escalado vertical`.

Inconvenientes de la programación distribuida

- Aumenta la complejidad
- Se necesita software nuevo especializado

- Problemas derivados de las comunicaciones (perdidas, saturaciones, etc.)
- Problemas de seguridad, ataques DDoS

Ejemplos de utilización de la programación paralela y distribuida

- Estudios meteorológicos
- Estudios del genoma humano
- Modelado de la biosfera
- Predicciones sísmicas
- Simulación de moléculas

Ejemplo de programación paralela y distribuida

Búsqueda de inteligencia extraterrestre - Proyecto SETI (https://setiathome.berkeley.edu/sah_about.php)

1.2.4. Condiciones de Bernstein

Una vez que sabemos qué es un programa concurrente y las distintas arquitecturas hardware que pueden soportarlo, vamos a ver qué partes de un programa se pueden ejecutar de forma concurrente y cuáles no.

Si observamos el siguiente código, queda claro que la primera instrucción se debe ejecutar antes que la segunda para que el resultado sea siempre el mismo (para los mismos datos de entrada).

```java
x = x + 1;
y = x + 1;
```

Sin embargo, en un código como el siguiente el órden en el que se ejecuten las instrucciones no influye en el resultado final (valor de las variables). En este caso se pueden ejecutar las tres sentencias a la vez incrementando la velocidad de procesamiento.

```java
x = 1;
y = 2;
z = 3;
```

A.J. Bernstein definió unas condiciones para determinar si dos conjuntos de instrucciones S_i y S_j se pueden ejecutar concurrentemente.

Para poder determinar si dos conjuntos de instrucciones se pueden ejecutar concurrentemente, se definen en primer lugar los siguientes conjuntos

- $L(S_k) = \{a_1, a_2, a_3, ...\}$ como el conjunto de lectura formado por todas las variables cuyos valores se leen durante la ejecución de las instrucciones del conjunto k.
- $E(S_k) = \{b_1, b_2, b_3, ...\}$ como el conjunto de escritura formado por todas las variables cuyos valores se actualizan durante la ejecución de las instrucciones del conjunto k.

Para que dos conjuntos de instrucciones S_i y S_j se puedan ejecutar concurrentemente, se deben cumplir estas tres condiciones de forma simultánea.

- $L(S_i) \cap E(S_j)$
- $E(S_i) \cap L(S_j)$

- $E(S_i) \cap E(S_j)$

Cuales de estas instrucciones se pueden ejecutar de forma concurrente

```java
a = x + y;
b = z - 1;
c = a - b;
w = c + 1;
```

Primero deberíamos obtener los conjuntos L y E para cada sentencia

> $L(S_1) = \{x, y\}$
> $E(S_1) = \{a\}$

> $L(S_2) = \{z\}$
> $E(S_2) = \{b\}$

> $L(S_3) = \{a, b\}$
> $E(S_3) = \{c\}$

> $L(S_4) = \{c\}$
> $E(S_4) = \{w\}$

Y ahora aplicarlas entre cada par de sentencias

> $L(S_1) \cap E(S_2) = \emptyset$ $E(S_1) \cap L(S_2) = \emptyset$ $E(S_1) \cap E(S_2) = \emptyset$ // Sí se pueden ejecutar concurrentemente

> $L(S_1) \cap E(S_3) = \emptyset$ $E(S_1) \cap L(S_3) = \{a\} \neq \emptyset$ $E(S_1) \cap E(S_3) = \emptyset$ // NO se pueden ejecutar concurrentemente

> $L(S_1) \cap E(S_4) = \emptyset$ $E(S_1) \cap L(S_4) = \emptyset$ $E(S_1) \cap E(S_4) = \emptyset$ // Sí se pueden ejecutar concurrentemente

> $L(S_2) \cap E(S_3) = \emptyset$ $E(S_2) \cap L(S_3) = \{b] \neq E(S_2) \cap E(S_3) = \emptyset$ // NO se pueden ejecutar concurrentemente

> $L(S_2) \cap E(S_4) = \emptyset$ $E(S_2) \cap L(S_4) = \emptyset$ $E(S_2) \cap E(S_4) = \emptyset$ // Sí se pueden ejecutar concurrentemente

> $L(S_3) \cap E(S_4) = \emptyset$ $E(S_3) \cap L(S_4) = \{c\} \neq \emptyset$ $E(S_3) \cap E(S_4) = \emptyset$ // NO se pueden ejecutar concurrentemente