I. Conceptos teóricos

Empezar con un capítulo titulado “Conceptos teóricos” quizá no es el mejor reclamo en un libro sobre programación, pero me temo que adquirir una base teórica es imprescindible antes de ponerse a escribir líneas de código. Conocer los conceptos que se explican en este capítulo nos permitirá asentar los fundamentos básicos que nos ayudarán a entender mejor el resto de los capítulos y nos permitirá también expresarnos con propiedad.

En las siguientes páginas, hablaremos sobre qué son los programas y procesos. Qué diferencias hay entre los términos procesador, microprocesador y núcleo. Cómo podemos clasificar los sistemas en función de su capacidad de trabajar o no con múltiples procesos de forma simultánea. Veremos también los fundamentos sobre cómo se gestionan los procesos en los sistemas multitarea. Finalmente, trataremos aspectos relacionados con la programación concurrente, el uso de hilos, la programación paralela y la distribuida. Que no cunda el pánico, será rápido y espero que indoloro.

Programas y procesos

Empezaremos con dos términos sencillos que, aunque utilizamos de forma indistinta, siendo estrictos son muy diferentes: Programa y proceso.

Un **programa** es un conjunto de instrucciones almacenadas en un fichero. Sin embargo, un **proceso** es una instancia de un programa en ejecución, es decir, el proceso se crea cuando ejecutamos un programa. Mientras que las instrucciones y los datos de los programas se almacenan en memoria secundaria (disco duro), las instrucciones y datos de un proceso se encuentran en la memoria principal (memoria RAM) o en los registros internos de la CPU.

Que ejecutemos un programa no implica que se cree un único proceso ya que se pueden lanzar múltiples. Por ejemplo, el navegador *Chrome* crea un proceso por cada pestaña que abrimos.

Procesadores, microprocesadores y núcleos

Procesador, microprocesador y núcleo son tres términos que muchas veces también se utilizan de forma indistinta. Veamos en qué se diferencian.

Como ya sabrás, el componente físico del ordenador que se encarga de ejecutar las instrucciones de los procesos es el **procesador o CPU** (*Central Processing Unit* o Unidad Central de Proceso). Un procesador se compone de varios elementos: unidad de control, unidad aritmeticológica, registros, etc. Al ser componentes tan complejos originalmente se construían mediante diferentes chips de forma que cada uno de ellos realizaba una tarea diferente.

En 1971 se desarrolló el primer **microprocesador**, que no era más que la implementación de un procesador o CPU en un único chip. Sin embargo, hoy en día utilizamos los dos términos, procesador y microprocesador, de forma indistinta.

Si analizamos las características de cualquier microprocesador actual, veremos que incluyen varios procesadores o CPU que reciben el nombre de **núcleos**. Los diferentes núcleos de un procesador permiten que se puedan ejecutar múltiples procesos de forma simultánea.

Sistemas monoproceso y multiproceso

En función de la capacidad que tiene un sistema para trabajar con diferentes procesos, hablamos de sistemas monoproceso o multiproceso.

Los **sistemas monoproceso** son aquellos en los que únicamente se puede ejecutar un proceso a la vez. Es decir, estos sistemas no permiten que en un mismo instante de tiempo se estén ejecutando dos procesos diferentes.

Por ejemplo, los microprocesadores *Intel Pentium* disponían de un única CPU, por lo que eran microprocesadores monoproceso. El sistema operativo *Windows 95* también era un sistema monoproceso porque no soportaba la ejecución de varios procesos de forma simultánea.

Figura 1: A la izquierda procesador Intel Pentium 4. A la derecha logotipo de Windows 95.

Los **sistemas multiproceso** son aquellos que permiten la ejecución de múltiples procesos de forma simultánea. Es decir, en un instante determinado de tiempo pueden existir múltiples procesos en ejecución. Los procesadores que disponen de múltiples núcleos son sistemas multiproceso. Como hemos mencionado anteriormente, un núcleo es una CPU, por lo que un microprocesador de 4 núcleos tiene integradas 4 CPU en su interior, permitiendo la ejecución de hasta 4 procesos de forma simultánea.

Sistemas monotarea y multitarea

Veamos una nueva forma de clasificar los sistemas. Según su capacidad para alternar la ejecución de procesos podemos hablar de sistemas monotarea y multitarea.

Los primeros ordenadores eran **sistemas monotarea**, donde el procesador no podía intercalar la ejecución de diferentes procesos. Hasta que no finalizaba la ejecución de un proceso no podía iniciarse la ejecución

del siguiente. De hecho, en la memoria de estos sistemas se alojaba un único proceso, el que se encontraba en ejecución en ese momento. En estos sistemas se infrautilizaba el procesador, ya que durante las ráfagas de E/S de los procesos el procesador quedaba ocioso, es decir, a la espera de más instrucciones para su ejecución.

¿Qué son las ráfagas de E/S?

Los periodos de tiempo en los que las instrucciones de un proceso se ejecutan en la CPU reciben el nombre de ráfagas de CPU. Sin embargo, hay períodos en los que el proceso no puede continuar su ejecución. Esto ocurre cuando realiza una petición de entrada / salida como puede ser la lectura o escritura de un archivo o la comunicación con un periférico. Puesto que los periféricos, discos duros, memoria principal y en general cualquier componente del ordenador es mucho más lento que el procesador, éste se ve obligado a esperar, quedando ocioso. Estos periodos reciben el nombre de ráfagas de entrada/salida o E/S.

En la siguiente figura se muestra la ejecución de un proceso en un sistema monotarea. Como se puede ver, cuando se produce una ráfaga de E/S el procesador queda ocioso.

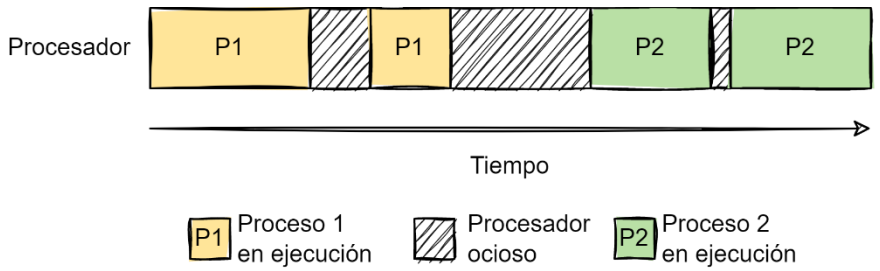

Figura 2: Sistema monoprocesador monotarea.

Para solucionar este problema se desarrolló en los años 60 una técnica denominada **multiprogramación**. Esta técnica permite alojar en la memoria principal múltiples procesos y ejecutarlos de forma intercalada, dando lugar a los sistemas multitarea.

En los **sistemas multitarea** varios procesos pueden intercalar su ejecución: Como se puede ver, en la siguiente figura, cuando un proceso se encuentra en una ráfaga de E/S, el sistema operativo pasa a ejecutar otro proceso. La ejecución de los procesos se intercala a tal velocidad, que da la impresión de que se ejecutan en paralelo (simultáneamente), aunque en realidad lo hacen uno detrás de otro. Gracias a esta técnica se aprovecha mejor la CPU, ya que se reduce el tiempo que permanece ociosa.

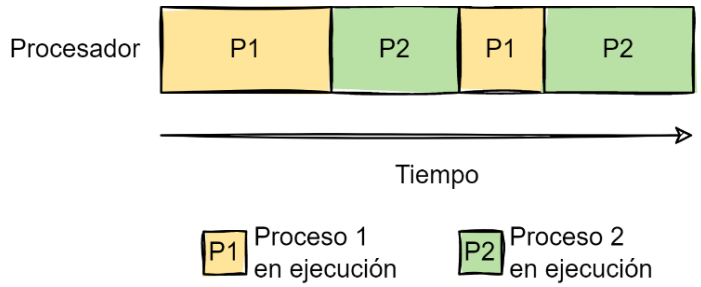

Figura 3: Sistema monoprocesador multitarea.

Hay que tener en cuenta que un sistema monoproceso puede ser monotarea o multitarea. Si es monotarea utilizará un único procesador y no podrá ejecutar el siguiente proceso hasta que no finalice el anterior. Si es multitarea utilizará un único procesador, que será capaz de intercalar en su ejecución diferentes procesos, aunque nunca ejecutaría dos procesos de forma simultánea.

De la misma forma, un sistema multiproceso puede ser monotarea o multitarea. Si es monotarea cada procesador podrá ejecutar un proceso diferente (cada proceso se ejecutará simultáneamente), pero hasta que no se finalice la ejecución de cada proceso, no se podrá ejecutar el siguiente. Si es multitarea cada procesador podrá intercalar en su ejecución diferentes procesos.

Para entender mejor todos estos conceptos utilizaremos un paralelismo con la cocina de un restaurante. La elaboración de un plato del menú requiere que el cocinero (procesador) realice una serie de pasos (tareas o procesos). Por ejemplo, para cocinar un plato de pasta a la boloñesa, hay que sofreír la carne, cortar la cebolla, hervir la pasta, etc.

Un **sistema monoprocesador y monotarea** equivaldría a tener un único cocinero que es incapaz de realizar varias tareas a la vez. Primero pone la pasta a hervir y se queda mirando a la espera de que termine la cocción. Cuando finaliza, pasa a realizar la siguiente tarea: sofreír la carne. De nuevo, hasta que la carne no está sofrita, no pasa al siguiente paso. Es evidente que estos sistemas son poco eficientes. El sistema mostrado en la Figura 2, aparte de ser monotarea, es monoprocesador, ya que dispone de un único procesador. Un ejemplo de sistema operativo monoprocesador y monotarea es el sistema operativo *MSDOS*.

Un **sistema monoprocesador y multitarea** equivaldría a tener un único cocinero que es capaz de realizar varias tareas a la vez. Primero pone la pasta a hervir, y mientras se calienta el agua, pone la carne a sofreír y corta la cebolla. El sistema mostrado en la Figura 3, aparte de ser multitarea, es monoprocesador, ya que dispone de un único procesador. Un ejemplo de sistema operativo monoprocesador y multitarea es *Windows 3.11* y *Windows 95*. En estos sistemas operativos el usuario ya podía realizar varias tareas como escuchar música mientras escribía un documento de texto.

Un **sistema multiprocesador y monotarea** equivaldría a tener varios cocineros que son incapaces de realizar varias tareas a la vez. Así, un cocinero pone la pasta a hervir y espera a que se cueza. Mientras tanto, el otro cocinero sofríe la carne y también queda a la espera. Hasta que no terminen sus pasos, ninguno de los dos cortará la cebolla. En la Figura 4 se puede ver la representación de varios procesos ejecutándose en un sistema multiprocesador y monotarea.

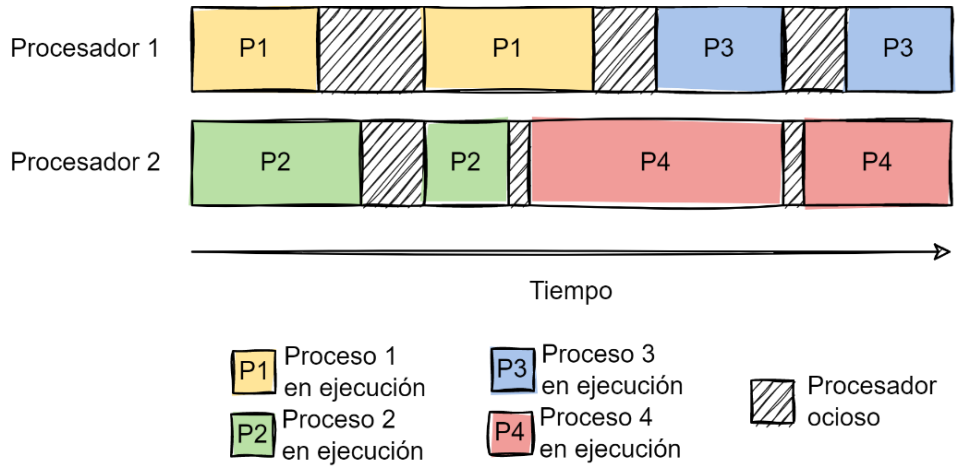

Figura 4: Sistema multiprocesador y monotarea.

Un sistema multiprocesador y multitarea equivaldría a tener varios cocineros que son capaces de realizar varias tareas a la vez. Así, un cocinero es capaz de hervir la pasta y sofreír la carne, mientras el otro corta la cebolla y realiza cualquier otra tarea. En la Figura 5 se representa la ejecución de procesos en un sistema de este tipo.

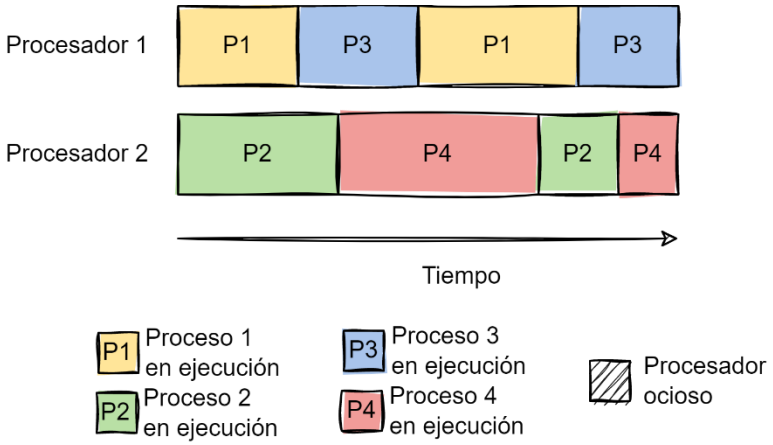

Figura 5: Sistema multiprocesador y multitarea.

Cualquier ordenador de sobremesa, portátil o teléfono móvil actual es multiproceso y multitarea. Podríamos pensar que, hoy en día, todos los sistemas son multiproceso y multitarea, pero hay que tener en cuenta que los procesadores están en todas partes ¡Hasta en las tostadoras! Y no siempre son sistemas que necesitan ejecutar una gran cantidad de procesos.

Gestión de procesos en sistemas multitarea

Ahora bien, ¿Cómo gestiona el sistema operativo la ejecución de los procesos en los sistemas multitarea? Sin duda es una tarea compleja, así que daremos una visión general. Las principales tareas que lleva a cabo el sistema operativo para gestionar los procesos son:

- **Creación de procesos**: Cuando se ejecuta un programa carga las instrucciones y datos en memoria RAM. Para cada proceso crea un registro denominado Bloque de Control de Proceso (BCP), donde se almacena toda la información relativa al proceso.
- **Eliminación de procesos**: Cuando la ejecución de un proceso finaliza, el sistema operativo libera la memoria y los recursos utilizados por el proceso.
- **Planificación de la ejecución de los procesos**: Decide qué proceso se ejecuta en cada momento. Esto ocurre, por ejemplo, cuando un proceso se encuentra en una ráfaga de E/S. Para que el procesador no quede ocioso, el sistema operativo desaloja el proceso y elige uno de los procesos en espera para que se ejecute. La elección de cuál será el siguiente proceso que se ejecutará la realiza el **planificador de procesos**.
- **Cambios de contexto**: Una vez elegido cuál será el siguiente proceso que se ejecutará, hay que desalojar el que se está ejecutando actualmente, guardar sus datos en el BCP y cargar el BCP del nuevo proceso.
- **Asignación de recursos:** Los procesos requieren el uso de memoria principal, acceso a archivos y el uso de periféricos. El sistema operativo se encarga de reservar estos recursos y asignarlos al proceso.

Bloque de Control de Proceso

Como hemos mencionado en el apartado anterior, el sistema operativo utiliza un registro denominado Bloque de Control de Proceso (BCP) para almacenar toda la información relativa del proceso. En este registro se almacena, entre otros datos:

- **Identificador del proceso**: Número que identifica de forma unívoca al proceso.
- **Estado del proceso**: Indica el estado en el que se encuentra el proceso. Veremos más adelante cuáles son estos estados.
- **Contador de programa**: Indica la dirección de memoria de la siguiente instrucción a ejecutar.
- **Registros de la CPU**: Valor de los registros de la CPU.
- **Información de planificación de la CPU.**
- **Información de gestión de memoria.**
- **Información sobre la entrada / salida**: Dispositivos asignados, archivos abiertos, etc.

Estados de un proceso

Ya hemos visto que los procesos se crean, se ejecutan, pueden permanecer a la espera o ser finalizados, es decir, pasan por diferentes etapas en lo que se llama su ciclo de vida. Estas etapas son:

- **Nuevo**: El proceso acaba de ser creado.
- **Preparado**: El proceso está listo para ser ejecutado, permanece a la espera de que el sistema operativo le asigne CPU.

- **Ejecución**: Sus instrucciones se encuentran en ejecución en el procesador.
- **Bloqueado**: El programa no ha terminado de ejecutarse, pero permanece a la espera de que ocurra un evento externo como puede ser la finalización de una operación de entrada / salida.
- **Finalizado**: El programa ha terminado su ejecución.

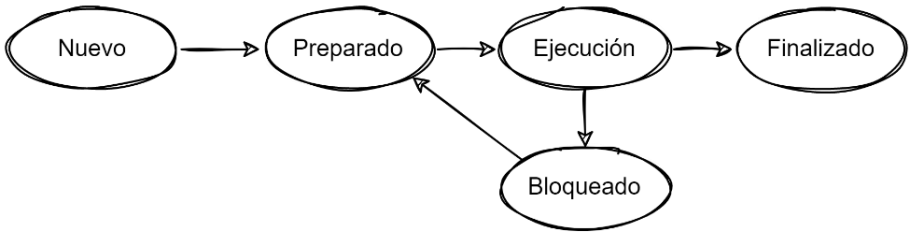

Figura 6: Estados de un proceso.

Hilos

Cuando se produce un cambio de contexto entre procesos, es decir, cuando se desaloja un proceso de la CPU para ejecutar otro diferente, el sistema operativo debe guardar, entre otras cosas, los valores de todos los registros de la CPU, información sobre el estado del proceso o los recursos utilizados, en el BCP del proceso y cargar los del nuevo proceso a ejecutar. Esta operación es costosa computacionalmente.

Además, cada proceso tiene su propio espacio de memoria principal reservado, sus propias variables y recursos asignados, por lo que compartir recursos, comunicar y sincronizar procesos no es una tarea sencilla o al menos todo lo sencilla que podría ser.

Los **hilos**, *threads* en inglés, procesos ligeros o también llamados subprocesos, solucionan estos problemas. Se tratan de secuencias de instrucciones de un mismo proceso que se pueden ejecutar de manera alternada o simultánea. Una de las diferencias con los procesos es que los procesos viven en el sistema operativo mientras que los hilos son partes de un proceso.

En la siguiente figura se puede ver una sencilla representación de la relación que existe entre procesos e hilos. Los procesos pueden tener de uno a múltiples hilos o subprocesos.

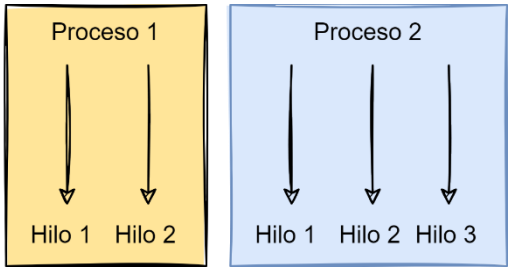

Figura 7: Procesos e hilos.

Los hilos no son entidades tan complejas como los procesos y por tanto se pueden gestionar de una forma más sencilla. Para empezar, los hilos de un mismo proceso comparten el mismo espacio de memoria, variables y recursos, por lo que el cambio de ejecutar un hilo a otro no es una operación tan costosa como el cambio de contexto entre procesos. Además, la comunicación entre hilos se simplifica, ya que se puede hacer a través de los espacios de memoria compartida.

Que un sistema sea multitarea permite que tengamos varios procesos abiertos. Permite, por ejemplo, que podamos escuchar música en *Spotify* mientras escribimos un documento en *Word* y el *Antivirus* hace un análisis del sistema.

Que un proceso tenga hilos permite que un mismo proceso pueda hacer varias tareas a la vez. Siguiendo los ejemplos anteriores, permite que *Spotify* reproduzca música mientras buscas las canciones de *J Balvin*, o que en el *Word* puedas escribir mientras el corrector ortográfico corrige las faltas.

Los hilos se utilizan mucho. Solo hay que pensar que todo programa que permita hacer más de una cosa a la vez hace uso de hilos. De hecho, cualquier programa con interfaz gráfica destina como mínimo un hilo solo para el dibujado de la interfaz y otro para toda la lógica del programa. En los siguientes capítulos nos centraremos en cómo programar utilizando hilos. De momento nos quedan unos pocos conceptos más que estudiar.

Programación concurrente

Vamos ahora con el concepto que da título a este libro. Según la *Wikipedia* el término **concurrente** hace referencia a la "habilidad de distintas partes de un programa, algoritmo, o problema de ser ejecutado en desorden o en orden parcial, sin afectar al resultado final". Así pues, programación concurrente hace referencia al hecho de desarrollar programas cuyas partes se pueden ejecutar de forma desordenada y esto se consigue mediante el uso de hilos.

Volvamos al ejemplo de la cocina. Imaginemos que nuestro cocinero (nuestro programa) quiere preparar un delicioso plato de pasta a la carbonara. Cada tarea representará un hilo: hervir la pasta, cortar la cebolla, batir las yemas, etc. El cocinero es capaz de hacer varias cosas a la vez, pero tiene que hacerlo con sentido. No debe mezclar las yemas con la pasta si ésta aún no está cocida. Aunque hay tareas que no requieren realizarse en un orden específico hay otras que sí.

El sistema operativo actúa de jefe, pero de jefe incompetente. Él es quien decide qué tarea se ejecuta en cada momento en la CPU. El problema es que el sistema operativo no sabe nada de cocina. No podemos culparle, tiene muchos trabajadores y no puede saber de todo. Él solo sabe que tiene un trabajador que quiere hacer varias tareas, pero no tiene ni idea de cuáles son ni en qué orden se deben realizar. Ante esta situación el sistema operativo no se complica la vida, ejecutará las tareas de forma desordenada a no ser que su trabajador le diga otra cosa.

Por eso, somos nosotros los que, al utilizar hilos, debemos utilizar mecanismos, llamados de comunicación y sincronización entre hilos, que aseguren que las partes importantes se ejecutan en el orden adecuado. El cocinero es el responsable de saber en qué orden debe realizar las tareas.

Como conclusión podemos definir la **programación concurrente** como el uso de técnicas para implementar algoritmos concurrentes, es decir, cuyas partes pueden ejecutarse de forma desordenada sin que esto afecte al resultado final. Como hemos dicho anteriormente, para ello se utilizan **mecanismos de comunicación** y **sincronización** que, por supuesto, estudiaremos más adelante.

Ventajas e inconvenientes

Las **ventajas** de la programación concurrente son:

- Se produce un **mayor aprovechamiento del uso de la CPU**, ya que mientras un proceso se encuentra bloqueado es posible ejecutar otro proceso.
- Conseguimos un **menor tiempo de ejecución**. En el caso de los sistemas multitarea se debe al mejor aprovechamiento del uso de la CPU. En el caso de los sistemas multiproceso se debe a que es posible ejecutar múltiples procesos de forma simultánea.
- **Facilita la resolución de ciertos problemas**. Imaginemos un servidor web que necesita responder a las peticiones de múltiples clientes. Si el tratamiento de las peticiones no fuera concurrente, el servidor no procesaría la última petición hasta haber respondido las anteriores.

Entre los inconvenientes encontramos solo uno, aunque bastante importante:

- La **programación es más compleja** ya que requiere la comunicación y sincronización de los procesos o hilos.

Aun así, las ventajas que aporta la programación concurrente supera con creces a sus inconvenientes.

Programación paralela y distribuida

Aquí tenemos otro par de conceptos que muchas veces se utilizan de forma indistinta cuando son cosas bien diferentes.

La **programación paralela** es un tipo de programación concurrente que se produce en los sistemas multiproceso donde múltiples procesos o hilos ejecutan sus instrucciones de forma simultánea. Esto es posible gracias al uso de varios procesadores o núcleos. Estos procesos trabajan de forma sincronizada para resolver una tarea compleja.

La **programación distribuida** es una forma de programación paralela que se basa en el uso de múltiples máquinas o nodos conectados a través de una red. De esta forma, es posible ejecutar de forma paralela los procesos, pero en máquinas que pueden estar ubicados en diferentes puntos. La comunicación en estos casos se suele llevar a cabo mediate *sockets*.

Escalabilidad vertical y horizontal

Si disponemos de una única máquina y los problemas que queremos resolver no dejan de aumentar en complejidad, necesitaremos mejorar sus recursos *hardware* (mejorar los procesadores, memoria *RAM*, etc.) para que tenga suficiente capacidad de cómputo.

Es lo mismo que ocurre cuando queremos jugar a un videojuego de última generación en un ordenador antiguo. Lo más seguro es que tengamos que comprar una nueva tarjeta gráfica, procesador o memoria *RAM*.

El hecho de aumentar la potencia de una máquina, mejorando o ampliando su *hardware* recibe el nombre de **escalado vertical**. Las ventajas de este tipo de escalado es que no tiene ninguna implicación sobre el *software*, que seguirá funcionando de la misma forma. El principal inconveniente es el incremento exponencial de los precios conforme mejoramos los componentes y que existe un límite tecnológico.

Una alternativa más económica y escalable es disponer de una red distribuida de ordenadores. Si queremos aumentar la capacidad de cómputo, solo será necesario añadir nuevos nodos. Este tipo de escalado recibe el nombre de **escalado horizontal**.

Su ventaja es que su capacidad de crecimiento es mayor, facilita la alta disponibilidad (si un nodo falla el sistema *no cae*, ya que el resto de los nodos siguen trabajando) y permite el balanceo de carga entre los nodos.

Entre sus inconvenientes encontramos que su configuración es más compleja y que requiere un *software* que coordine los diferentes nodos.