Multitarea e Hilos, fácil y muchas ventajas

La descripción que te dispones a leer, pese a su carácter sencillo e ilustrativo tiene un alto contenido informativo. El autor considera que, lo aquí abajo escrito, pueda que no sea comprensible para quien todavía no tenga clara la orientación a objetos (recomiendo leer antes este otro tutorial). Y sin más dilación…
Existe una ley informática llamada ley de Wirth que reza (extraída de http://es.wikipedia.org/wiki/Ley_de_Wirth):

> *"El software se ralentiza más deprisa de lo que se acelera el hardware"*

Y sintiéndolo mucho, tengo que darle la razón, porque los programadores de aplicaciones, en la mayoría de los casos, no sabemos emplear la potencia del hardware y no se optimiza. Este artículo intentará solventar una gran parte.

Hacer varias cosas de manera simultánea, que gran regalo y que lio ¿de verdad? No, solo hay que entender en que se basa y como se programa.

¿Regalo? Puede que también caigas en la pregunta ¿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 (Seguro recuerdas cualquier pantalla de "cargando…" o en inglés "Loading…", ¿Seguro que era necesaria?); notar que nos referimos a la inutilidad de la mayoría de las veces de bloquear al usuario, no a la ejecución que se produce debajo de esa pantalla de carga.

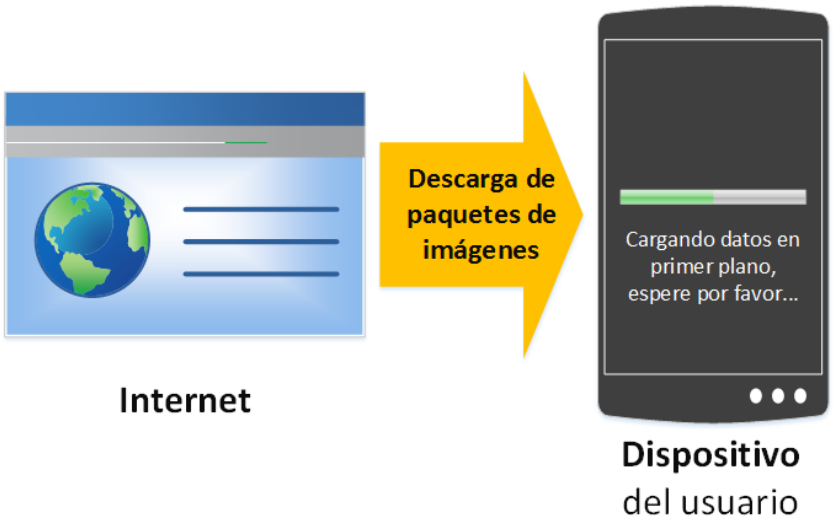
Otro efecto molesto y estarás de acuerdo conmigo, es aquel que se da cuando el usuario pulsa un botón y la aplicación se bloquea, pero no se bloque por un error, sino porque debajo se está ejecutando un montón de código que requiere tiempo de proceso para concluirse (sí, no te rías por recordar alguna que otra vez algún bucles for o while, que tardaba más de la cuenta en terminar y se notaba; lo hemos hecho todos). Todo esto se soluciona con la multitarea ¿Todo? Sí, pero puede que antes haya que pensar un poco para caer en la solución con ésta. Con la multitarea el usuario nunca quedará bloqueado -pudiendo 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. Claro está, si se hace bien.

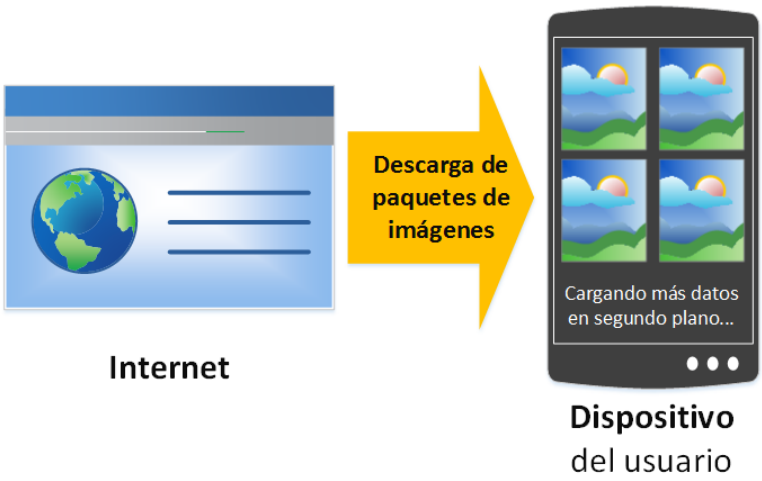
Es muy importante entender bien como se controla la multitarea, pues es la única manera que tenemos de desgranar el código de una aplicación, y que se ejecute en menos tiempo o para no bloquear al usuario.

El concepto de **Hilos** (en inglés Thread) nos ayuda. 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? Vamos a poner el siguiente ejemplo, imagina que queremos ver un listado de 100 imágenes que se descargan desde Internet, como usuario ¿Cuál de las dos opciones siguientes elegirías?:

A) 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.



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



Como desarrollador la opción A es más sencilla ¿Seguro? Pero no te he preguntado cómo desarrollador, te he preguntado lo 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 decanta 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 nos llena de gozo.

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

- **Primer plano**: Aquí se ejecuta únicamente un hilo llamado "hilo principal". Aquí hemos programado siempre, sin conocimiento de que estábamos trabajando ya con hilos. Es el hilo que trabaja con las vistas, es decir, con la interfaz gráfica que ve el usuario: botones, ventanas emergentes, campos editables, etc. También, puede ser usado para hacer cálculos u otros procesamientos complejos, aunque estos deberían de evitarse hacerse en este hilo a toda costa –salvo si es imposible que se hagan en otro hilo. Cabe señalar, que el primer plano influirá en la felicidad del usuario con nuestra aplicación. Aquí es donde el usuario interacciona de manera directa, además todo lo que pase aquí lo ve y lo siente. El desarrollador ha de tener especial cuidado al trabajar con el hilo principal, pues será juzgado por el usuario –si la aplicación va lenta es porque el primer plano va lento y esto al usuario no le gusta nada. También es importante saber, que una mala gestión del primer plano por parte del desarrollador, será castigada por el sistema operativo (por ejemplo: en Android si el hilo principal de una aplicación es bloqueado más de 5 segundos, la aplicación se cerrará mostrando una ventana de forzar cierre; y seguro que recuerdas comportamientos parecidos en otros sistemas operativos cuando te dice que "la aplicación no responde, ¿deseas finalizar su ejecución?").

- **Segundo plano (o en inglés backgraund)**: Se ejecuta todo el resto de hilos. El segundo plano tiene la característica de darse en el mismo momento que el primer plano. Aquí los hilos deberían de llevar las ejecuciones pesadas de la aplicación. El segundo plano el usuario no lo ve, es más, ni le interesa, para el usuario no existe. Por lo que comprenderás, que el desarrollador puede moverse libremente por este segundo plano –dentro de unos límites. Aquí el desarrollador se puede resarcir y hacer que, por ejemplo, un método tarde horas en ejecutarse, ya que el usuario ni lo sentirá –aunque si está esperando sí que lo notará, con lo que un poco de cuidado también.

Lo principal es claro: no debemos interrumpir al usuario nunca. Por lo que: no debemos hacer cosas que consuman muchos recursos en el hilo principal, hilo que corre en primer plano. Realmente, una vez que entiendas al 100% cómo funcionan los hilos, casi todo nuestro programa debería de ejecutarse en hilos bien separados y cuanto más pequeños mejor.

Antes de continuar, vamos a notar una cosa que suele suscitar dudas: **no confundamos el término proceso con hilo**. Un proceso es el programa o aplicación en ejecución (Extiendo un poco más para que queden claras las diferencias. Lo que llamamos aplicación es el fichero ejecutable almacenado en memoria. Varios procesos pueden ejecutar varias instancias del mismo programa, es decir, como cuando se abren varias ventanas de un Bloc de notas o un Word). Así, se deduce y es verdad que un proceso contiene un hilo –mínimo el hilo principal que corre en primer plano- o varios hilos -El principal más algunos en segundo plano.

Para no liarnos con los hilos (vago chiste, lo sé), vamos a ver un ejemplo a modo de comic. Retomemos el ejemplo anterior: un listado de imágenes que se descargan desde Internet. Para hacerlo más cómodo y comprensible, vamos a ejemplificarlo con personas. Una persona es un hilo, como el siguiente:

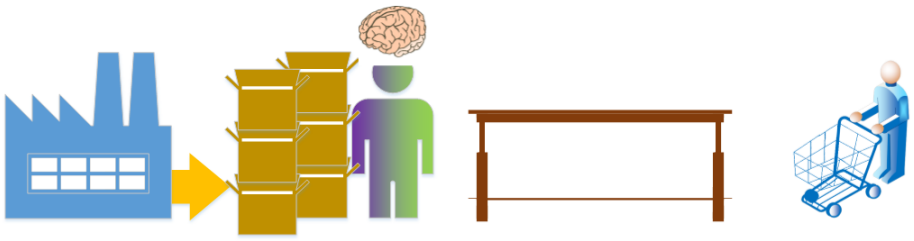
Hilo que lo hace todo
encargado de descargar
los datos de Internet
Y también encargado
dibujar las descargas en
la pantalla

Este honorable trabajador (nuestro hilo), hace su trabajo de la mejor manera que sabe, pero tiene demasiadas responsabilidades. Para lo que le han contratado es para llevar paquetes desde una fábrica hasta el escaparate de una tienda. Además, este trabajador/hilo es el hilo principal de la aplicación.

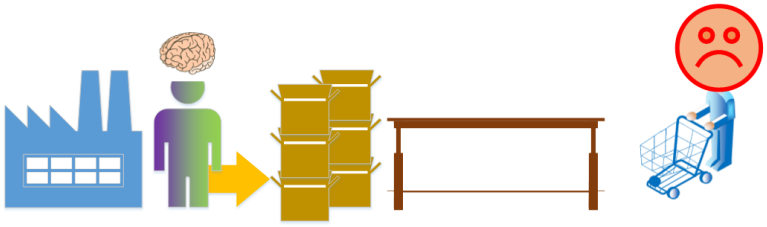
Veamos cómodamente, los pasos en modo de comic, de todo el proceso de este trabajador/hilo, con sus correspondientes equiparaciones en la vida de un programa. Un inciso antes de continuar, el trabajador tiene un cerebro ¿Ya lo sabías no? Lo sepas o no da igual, porque de momento no vas a hacer ni caso al cerebro en las siguientes imágenes, el significado del cerebro se revelará más adelante; de momento me conformo con que te creas que tiene un cerebro y al cerebro ni caso . A continuación el ejemplo, la palabras raras del título se explicarán también más adelante, es importante comprender primero como trabajan los hilos y luego el significado de las palabras raras vendrá solo. El ciclo completo de lo que este trabajador/hilo hace es:

EJEMPLO SIN CONCURRENCIA NI PARALELISMO, ES DECIR, SIN MULTITARÉA

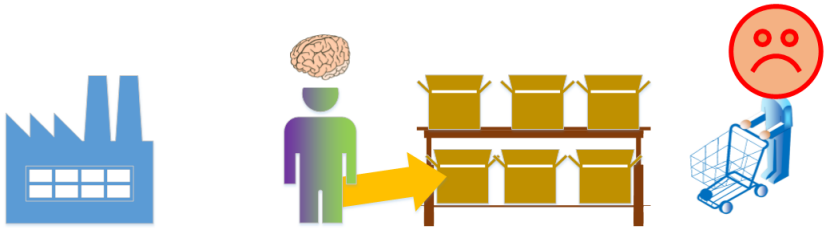
1) Primero busca todos los paquetes de datos con las “imágenes” de una fábrica que se llama “Internet”.



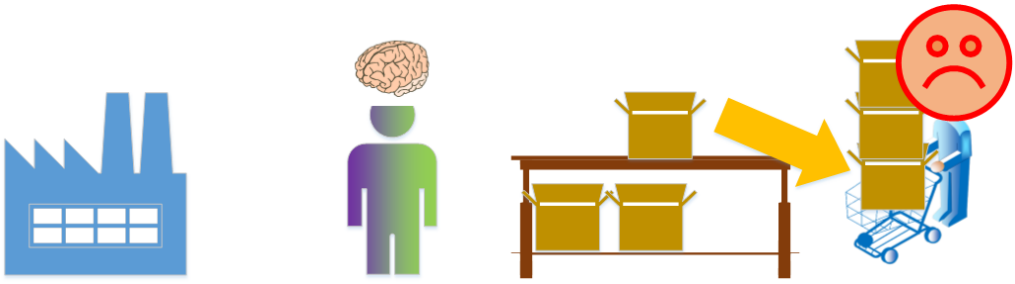
2) Lleva todos estos paquetes en camión y los descarga a una tienda llamada “dispositivo del usuario”.



3) Luego le toca colocar todo y cada uno de los paquetes que ha trasportado al escaparate de la tienda, que para aclarar también podemos llamar como “pantalla del dispositivo”.



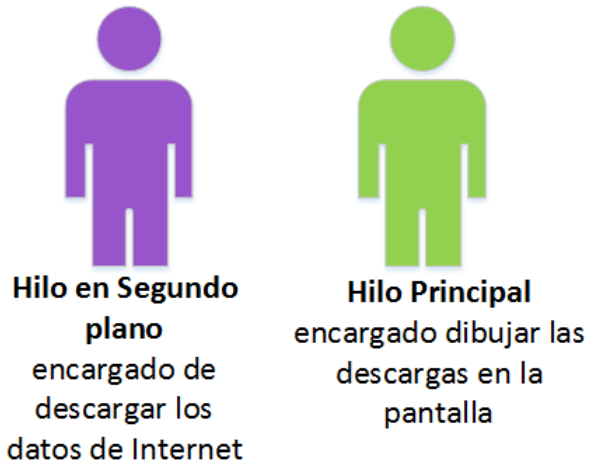
4) Todo esto se hace con la finalidad de que el cliente o “usuario” los consuma. Eso sí, solo en este momento se le permite al usuario poder consumirlos –ya que es el momento que tiene algo que consumir, antes no había nada- con lo que el resto del tiempo estuvo esperando.



¿Un emoticono triste, tirando a muy enfadado? Sí, el usuario lleva bastante tiempo enfadado con nuestra aplicación. Ha tenido que esperar demasiado y además cuando no hacía falta. Un usuario descontento se traducirá en menos estrellas en nuestra

aplicación, en malas críticas, o en otras maneras de expresar una opinión negativa hacia nuestro programa.

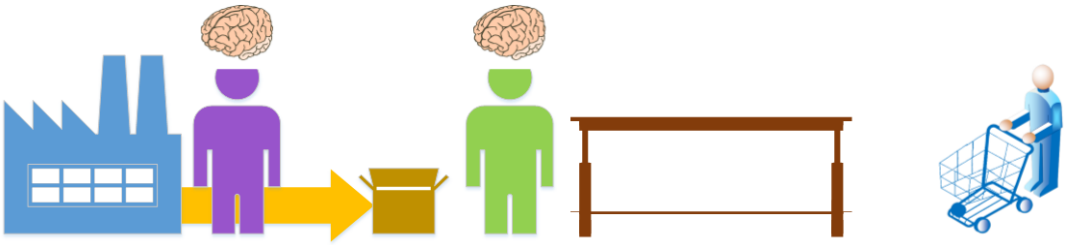
Seguro que te has fijado en el doble color de nuestro trabajador/hilo que es un hilo principal todo él. Tiene dos colores para representar la gran carga de trabajo que tiene este hilo. Habrá que investigar si se puede hacer de él dos hilos diferentes ¿Será necesario contratar a un segundo trabajador/hilo? Espero que respondas afirmativamente, pues es algo que podría hacerse en dos hilos diferentes, uno principal –encargado de dibujar en pantalla- y otro en segundo plano –encargado de descargar los datos desde Internet.



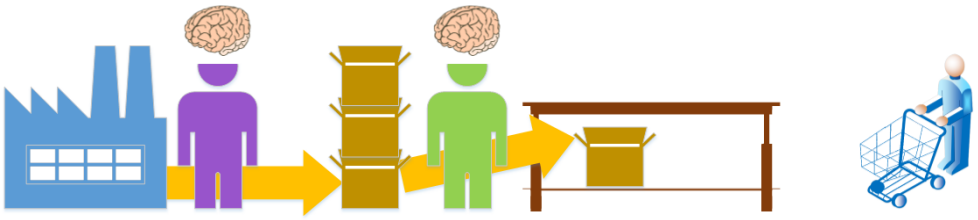
Teniendo dos trabajadores/hilos es lógico pensar que todo vaya mejor y más rápido (¡Para eso hemos contratado más manos de obra!). Dispondremos de un trabajador/hilo principal cuya dedicación es exclusiva para el usuario. Y otro hilo/trabajador en segundo plano, que se dedica a descargar los paquetes de datos, cuya función no la siente el usuario y nunca sabrá que está ahí trabajando para él. Vuelvo a notar en este ejemplo, de todavía no hacer caso a la imagen del cerebro, ya queda poco para hablar de este órgano de proceso (es una pista de qué tratará, pero no nos preocupa ahora, sino un poco más adelante). Por lo que la manera de procesar estos trabajadores/hilos será el siguiente:

EJEMPLO DE CONCURRENCIA Y PARALELISMO, LA MULTITARÉA EN TODO SU ESPLENDOR

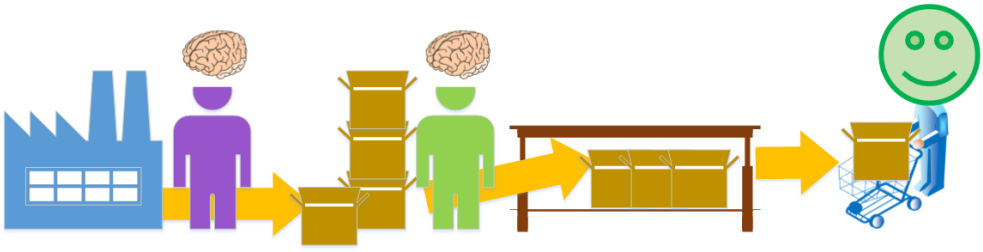
1) Para empezar, el trabajador/hilo en segundo plano toma algún paquete de datos de la fábrica que llamamos “Internet”, lo lleva en camión y lo descarga en la tienda. Donde el trabajador/hilo principal está esperando a que le llegue algún paquete para poder realizar su labor.



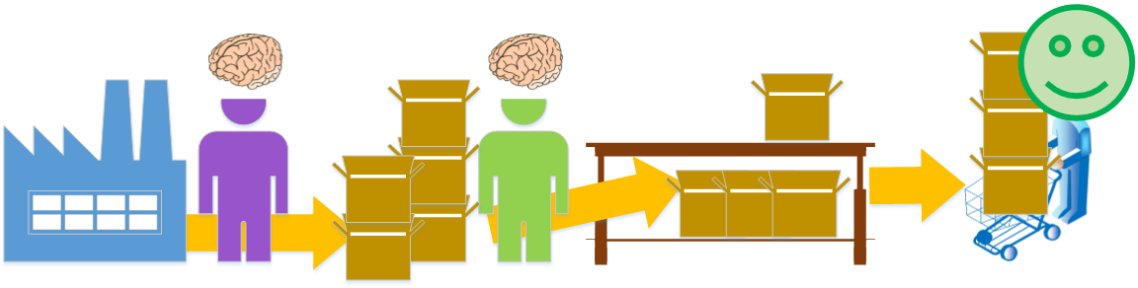
2) El trabajador/hilo principal ya tiene algo que hacer. Toma el paquete que le ha llegado y lo coloca en el escaparate de la tienda (que hemos llamado casualmente como "pantalla del dispositivo"). Mientras tanto, el trabajador/hilo en segundo plano no ha terminado su trabajo, con lo que continúa llevando paquetes a la tienda desde la fábrica. Cabe notar, que ha sido dicho y hecho, el cliente/usuario ya tiene al menos un paquete que consumir en la tienda.



3) El proceso continúa. El trabajador/hilo en segundo plano lleva a la tienda los paquetes, para que el trabajador/hilo en primer plano los coloque en el escaparate. Por lo que, el cliente/usuario puede ir consumiendo cuanto desee de lo que exista ya en el escaparate.



4) Esto durará hasta que se cumpla alguna condición de finalización o interrupción de algún trabajador/hilo, las cuales pueden ser muchas (por ejemplo: apagar el dispositivo, que se hayan acabado los paquetes que descargar, etc).



Con los hilos aparece la verdadera magia: la concurrencia ¿Concurrencia? Y con ello se posibilita la programación en paralelo ¿Paralelo? Ya empezamos con los términos raros de la multitarea en todo su esplendor. Tranquilo con las palabras extrañas, no hace falta que te las aprendas, pero al final te las terminaras sabiendo por repetición y por aplastante lógica.

Te acuerdas del primer ejemplo de la fábrica y la tienda (recordar el "Ejemplo sin concurrencia ni paralelismo, es decir, sin multitarea"), en el que solo había un hilo principal que lo hace todo. En este ejemplo tenemos que todo se hace en un tiempo X que se estima largo. Podemos representar por la siguiente barra de tiempo desde el principio de la ejecución del hilo hasta el fin:

En comparación con este segundo ejemplo, en el que ya tenemos dos hilos, uno principal y otro secundario, que trabajan de manera simultánea. Es deducible que el tiempo que tarda en hacerse todo el proceso es bastante menor, ya que se sobrexponen los tiempos de ambos hilos. En la siguiente representación vemos como el hilo principal se ejecuta de manera "paralela" al hilo en segundo plano y que es menos a la anterior barra de tiempo:

Esto nos concluye en la definición de **paralelo**: dos o más hilos que se ejecutan de manera simultánea.
¿Pero todos los hilos que pueden ejecutarse en paralelo se ejecutan todos a la vez? No siempre. Depende de un factor delimitador más a tener en cuenta, que aunque no lo controla el desarrollador de aplicaciones existe siempre –es más, al desarrollador de aplicaciones no le importa demasiado, pero conocerlo explica muchas cosas y es muy útil para entender bastante, por lo que en Jarroba.com te ayudaremos a entenderlo.

Inundémonos de un conocimiento que seguro ya hemos escuchado docenas, que digo docenas miles de veces: los núcleos de un procesador. O sí, cuántas veces nos habrán vendido un ordenador o móvil por los núcleos de su procesador; convenciéndonos de que así irá más rápido, adornado de la palabra "multitarea". ¡Qué decepción!, la mayoría de las aplicaciones no los aprovechan porque los desarrolladores de aplicaciones no los

han tenido en cuenta, no han sabido aprovechar la tecnología. Pero tú, atento lector, sabrás como aprovechar la última y máxima tecnología para tus futuros desarrollos.

Bueno, sabiendo que un **procesador** es el encargado de ejecutar los programas. Y que un procesador puede tener entre un núcleo y muchos.

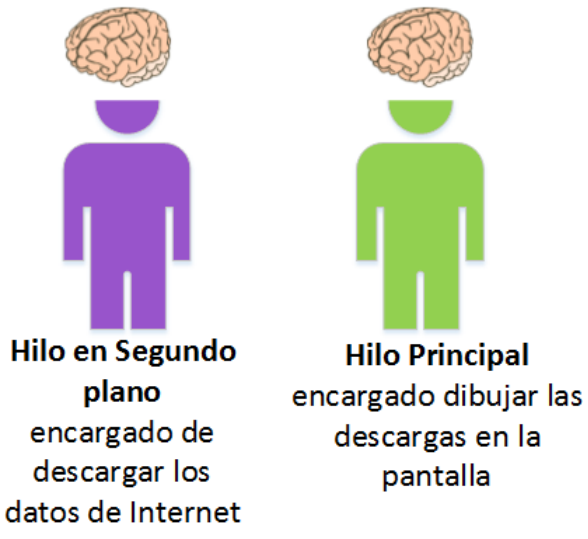
Ejemplo de 1 procesador que contiene 2 núcleos

Sirviendo los **núcleos** para procesar de manera simultánea varios hilos (he aquí otra vez el amigo hilo). Esto es, cada núcleo ejecuta un hilo a la vez. Por lo que si tenemos un procesador con dos núcleos, podremos ejecutar dos hilos a la vez. Si tenemos uno de cuatro núcleos podrá ejecutar un máximo de cuatro hilos de manera simultánea. Y así con todos los núcleos de los que dispongamos.
Para seguir con nuestro ejemplo, con los trabajadores que son hilos, el núcleo sería el cerebro de cada trabajador/hilo. Un trabajador/hilo siempre tendrá una misión que hacer determinada, pero sin un cerebro/núcleo no puede hacer nada. Y aquí ya sí que nos metemos de lleno con el órgano de proceso de nuestra cabeza, que equiparamos al núcleo de un procesador; no al procesador que puede tener varios núcleos y, por tanto, varios cerebros.

Así, si repasamos el primer ejemplo de la fábrica y la tienda con un solo trabajador/hilo, con un cerebro/núcleo (Ahora sí hacemos caso a los cerebros de las imágenes), le es más que suficiente para ejecutar todo de manera correcta. Le sobraría con un procesador con un núcleo, pues más núcleos no los aprovecharía.

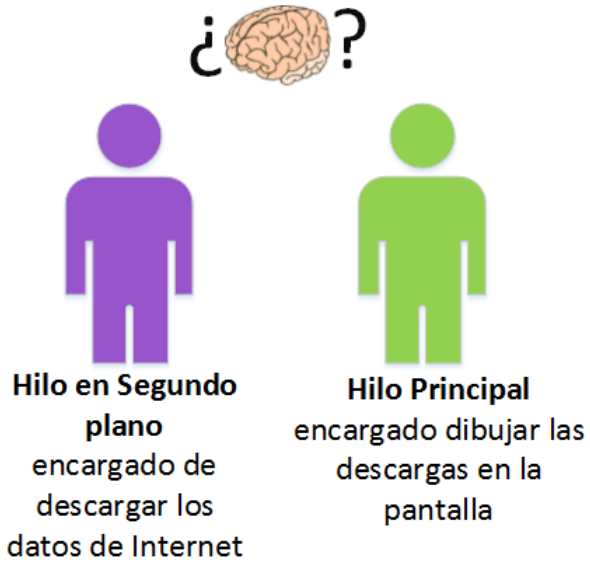
Por lo que un cerebro/núcleo procesará el trabajo tan rápido como pueda en el tiempo (Esta rapidez se mide con la velocidad de reloj de un núcleo de un procesador, es decir, los Hertzios; cuando escuchamos que tiene, por ejemplo, 3.4GHz, esta es la velocidad que tiene cada núcleo del procesador en procesar los datos que le llegan). Vuelvo a dibujar su gráfica de tiempo con un cerebro/núcleo al lado, para representar el tiempo que tarda este en procesarlo todo.

El segundo ejemplo es más complejo. Estamos suponiendo que tenemos dos núcleos. Si lo volvemos a repasar, fijándonos en los cerebros/núcleos, podemos llegar a la conclusión que existen dos núcleos, uno por cada hilo que se ejecuta.



Así, con dos cerebros/núcleos se puede trabajar en paralelo como indicamos anteriormente.

Pero también se puede dar lo siguiente con el segundo ejemplo ¿Qué ocurriría si tenemos un procesador con un único núcleo? ¿Qué hilo se ejecutaría? ¿Uno primero y otro después? ¿Dará error la aplicación? ¿Qué hará?



Es evidente que no hay suficientes cerebros/núcleos para tantos trabajadores/hilos. En algún momento uno de ellos no tendrá cerebro/núcleo y se tendrá que quedar idiota –detenido completamente- mientras el otro con cerebro/núcleo realiza sus labores. Lo razonable y justo es que compartan el cerebro/núcleo un rato cada uno para poder trabajar los dos.

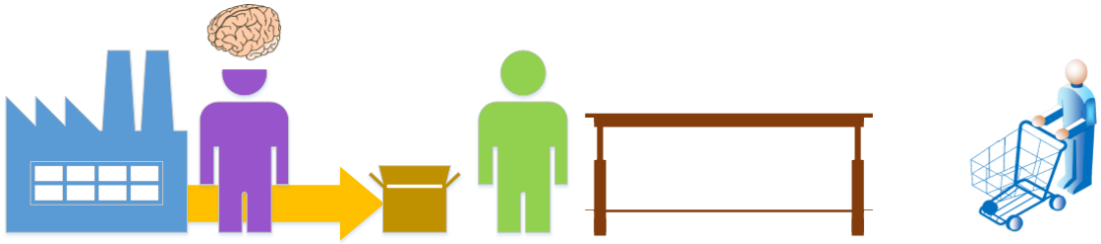
Y así es como se hace. El procesador se encarga de decidir cuánto tiempo va a usar cada hilo el único núcleo que existe. Dicho de otro modo, el procesador partirá los hilos en cachos muy pequeños y los juntará salteados en una línea de tiempo. Al procesarse los dos hilos de manera salteada, y gracias a la gran velocidad con la que procesan datos los procesadores hoy día, dará la sensación de paralelismo, pero no es paralelismo. Si nos fijamos en la siguiente línea de tiempo, la longitud es tan larga como si lo hiciéramos todo en un hilo.

No es paralelo pero sí es concurrente. La definición de **concurrencia**: dos o más componentes entre ellos independientes (por ejemplo, hilos con funcionalidades separadas) se ayudan para solucionar un problema común (en estos ejemplos, los hilos trabajan juntos para mostrar al usuario unos paquetes descargados de Internet). Lo concurrente se recomienda ejecutarse en paralelo para sacar el máximo partido de la concurrencia, pero como vemos no es obligatorio. Si volvemos la vista atrás, y repasamos el primer ejemplo con un solo hilo (repasar "Ejemplo sin concurrencia ni paralelismo, es decir, sin multitarea"), comprobaremos que no existe ni la concurrencia ni el paralelismo. Y en el segundo ejemplo, con dos hilos y dos núcleos (repasar "Ejemplo de concurrencia y paralelismo, la multitarea en todo su esplendor"), se da tanto la concurrencia como el paralelismo.
Estoy convencido de que te preguntarás ¿Si no existen varios núcleos en el procesador, entonces no tiene ninguna ventaja haber desarrollado la aplicación con hilos? Pero sí las tiene. Si lo ejecutamos todo en el hilo principal, siempre se va a bloquear al usuario. Pero si lo ejecutamos con hilos, aunque solo dispongamos de un núcleo, habrá en algún instante en el que el usuario tenga con que actuar, y el procesador ya se encargará de darle preferencia al usuario para no bloquearle.

Veamos el siguiente ejemplo en formato comic de los trabajadores con un solo cerebro/núcleo que han de compartir:

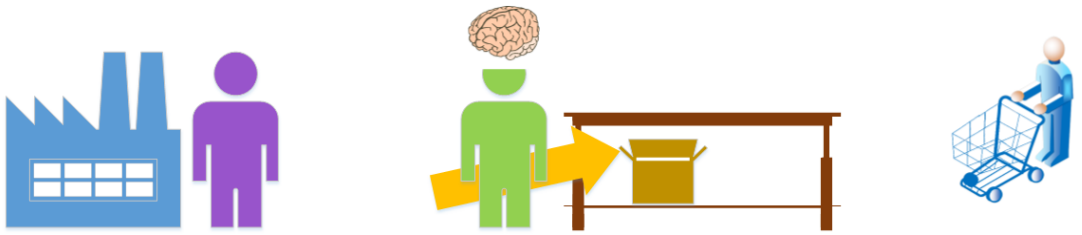
EJEMPLO DE CONCURRENCIA PERO NO DE PARALELISMO, ES MULTITARÉA AUNQUE NO LA IDEAL

1) El procesador decide que empiece poseyendo al cerebro/núcleo el trabajador/hilo en segundo plano, para que descargue algún paquete de datos. Mientras, el trabajador/hilo principal no podrá hacer nada porque no tiene cerebro.

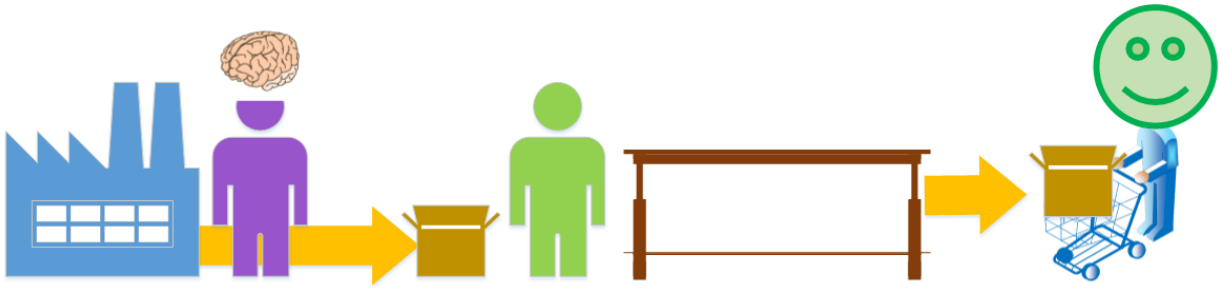


2) El procesador ha decidido que ya lo ha tenido mucho tiempo el trabajador/núcleo en segundo plano y le pasa el turno al trabajador/hilo principal. Este, mientras pueda usar al cerebro/núcleo, colocará todos los paquetes de datos que puedan en el

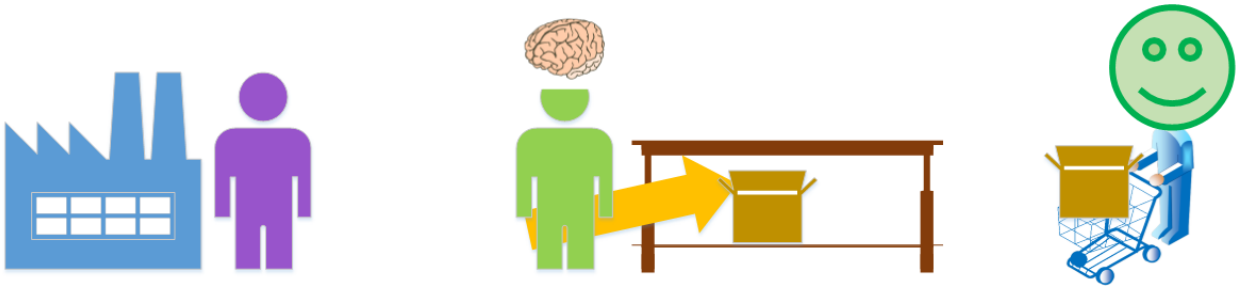
escaparate. Decir que el trabajador/hilo en segundo plano se ha quedado lelo y no puede hacer nada por no tener cerebro/núcleo.



3) El procesador dice: "Turno acabado". Le toca otra vez al trabajador/hilo secundario, que realiza su trabajo. El trabajador/hilo principal no hace nada. Y ya por lo menos el cliente/usuario dispone de algún dato.



4) Este vaivén de turnos cerebrales se repite.



Se aprecia en este ejemplo, que no es tan rápido como disponer de dos núcleos, pero tampoco tan lento como trabajar con un solo hilo que lo haga todo -en poco tiempo el usuario ya tiene algo que hacer.

En este ejemplo se ha simplificado el orden de turnos del procesador. En la mayoría de los casos, el procesador no espera a que se termine la acción completa antes de ofrecer el turno del núcleo a otro hilo. Es decir, el procesador no esperaría a que el hilo que descarga los datos, descargue el 100% del paquete entero para pasarle el núcleo al otro hilo. Sino que le puede quitar el núcleo al hilo secundario que descarga, por ejemplo, al 23,567% de la descarga. Y lo mismo con el hilo principal, podría quitarle el núcleo a mitad de la acción de colocar el paquete en el escaparate. Eso sí, al volver a tener el turno del procesador continuará desde donde se quedó. Las únicas acciones que no

puede dividir el procesador son aquellas sentencias que se hacen llamar **operaciones atómicas**: son aquellas que no son divisibles por el procesador y se tienen que ejecutar de manera completa.
Hemos hablado de varios hilos en un solo programa, pero no de la ejecución simultánea de varios programas. Por ello lanzo una pregunta sencilla, pero que espero sea aclarativa. Si tenemos dos programas diferentes, cuyos programadores no hicieron uso de los hilos porque no sabían los que eran: un reproductor de música y un navegador de Internet ¿Qué crees que ocurriría si quiero escuchar música mientras navego por Internet si se ejecuta todo en un procesador de dos núcleos? Piénsalo un poco, porque la respuesta es que escucharía la música fluida y sin cortes, además navegaría por Internet también sin cortes. Esto se debe a que cada programa –pese a que no se usaron hilos para su desarrollo por desconocimiento- tiene por lo menos cada uno un hilo principal. Por tanto, los dos programas -que cada uno es un hilo íntegramente- puede ejecutarse sin interrupciones, cada uno en un núcleo diferente. Como hemos indicado y a modo de resumen: cada programa tiene al menos un hilo, que es el principal. Si las aplicaciones tuvieran varios hilos y se ejecutaran a la vez, tampoco pasaría nada raro; ya que el procesador y el sistema operativo se encargan de gestionar el tiempo que estarán usando cada hilo los núcleos y se produciría la multitarea.

Como programadores ¿Cómo tenemos que actuar? Tenemos que actuar como si existieran en todo momento infinitos núcleos ¿Pero la tecnología no ha llegado a tanto? Bueno, se acercará en un futuro no muy lejano. Solo hay que ver que los procesadores tienen cada vez más núcleos, y cada nuevo que sale al mercado tiene el número núcleos a la potencia de dos del anterior más nuevo (los primeros procesadores tenían 1 núcleo, luego llegaron los de 2, luego los de 4, los de 8, 16, y así hasta el infinito). También tenemos que actuar como si solo existiera nuestra aplicación en ejecución; de dividir los recursos entre las diferentes aplicaciones ya se encarga el sistema operativo.

Dicho de otro modo, da igual cuantos núcleos tenga el dispositivo del usuario, como programadores debemos programar como si hubiera todos los que necesitamos, y como si solo existiera nuestra aplicación. Al fin y al cabo no nos influye le número de núcleos ni de otras aplicaciones al programar con hilos. De esto se preocupan los que diseñan y construyen: el procesador, el sistema operativo, entre otros. Pero sí nos tiene que preocupar el usuario, quien conoce o debería conocer las características de su dispositivo, de manera tal que sabe en todo momento que aplicaciones van bien y cuales desinstalará porque van mal.