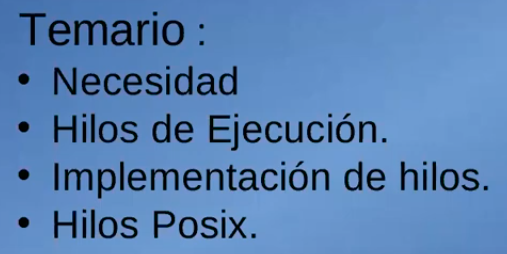
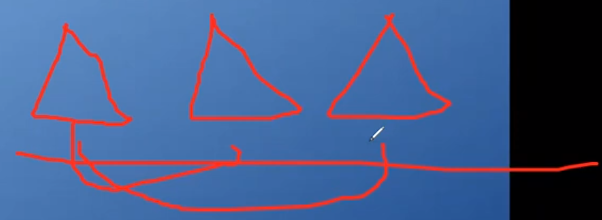
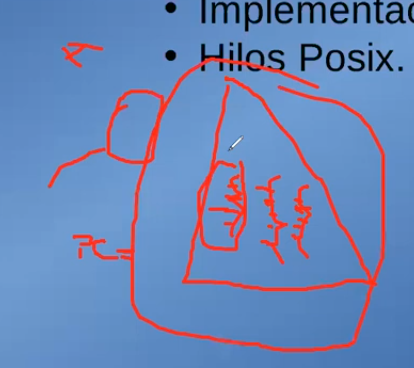
*-Clase del 21-04-2021 Hilos.*

**¿Por qué yo necesitaría usar hilos?**

Habíamos dicho que, si yo tengo que hacer una tarea, para acelerarla agarro y creo 2 o 3 procesos y que cada uno haga un pedacito de esa tarea y obviamente utilizando algún IPC intercambian información. Esto esta bien y me sirve para acelerar mi tarea.

La técnica de hilos sirve para lo mismo, acelerar la tarea que quiero hacer, pero el enfoque o aproximación, es bastante distinta de lo que hago con multiprocesos y una comunicación entre ellos.

Se basa en que un proceso está compuesto por un flujo de ejecución, o la ejecución de una receta o un programa y los recursos que tiene, lo que intenta hacer con hilos es, dentro de un programa crear mas de un flujo de ejecución de código. Lo que intenta decir es que dentro de un programa en ves de tener un solo contador del programa y ejecute la instrucción 100, 101, 102 etc. y asi… de alguna manera yo logro tener “n” de estos bloques de ejecución.

Esto significaría por ejemplo si dentro de un proceso voy a tener 2 hilos, necesitaría al menos 2 contadores de programa, porque cada hilo va a ejecutarse por su cuenta cuando el planificador le asigne CPU.

¿Qué son?

A partir de todo, podríamos definirlos, no como mini procesos, sino como la unidad básica de planificación.

Se creo la abstracción de proceso para aprovechar, cuando tengo varios programas y alguno se quedaba bloqueado esperando entrada salida, esta abstracción nos permite meter otro programa en memoria y ejecutarse. Esto es procesos

Ahora si pudiéramos ver como una unidad a un proceso y decir, cuando el proceso esta bloqueado, lo separo en distintas partes, y si alguna se puede seguir ejecutando, la sigo ejecutando. Lo que haría que mi tarea se ejecute mas rápido, porque cuando un proceso se bloquea, por ejemplo, en espera de teclado, si yo pudiera que otra parte del proceso se pudiera ejecutar independientemente eso seria aprovechar el tiempo ocioso de ese proceso.

Se dice que un proceso es a la CPU, lo que un hilo es a un proceso.

Ejemplo de un chat.

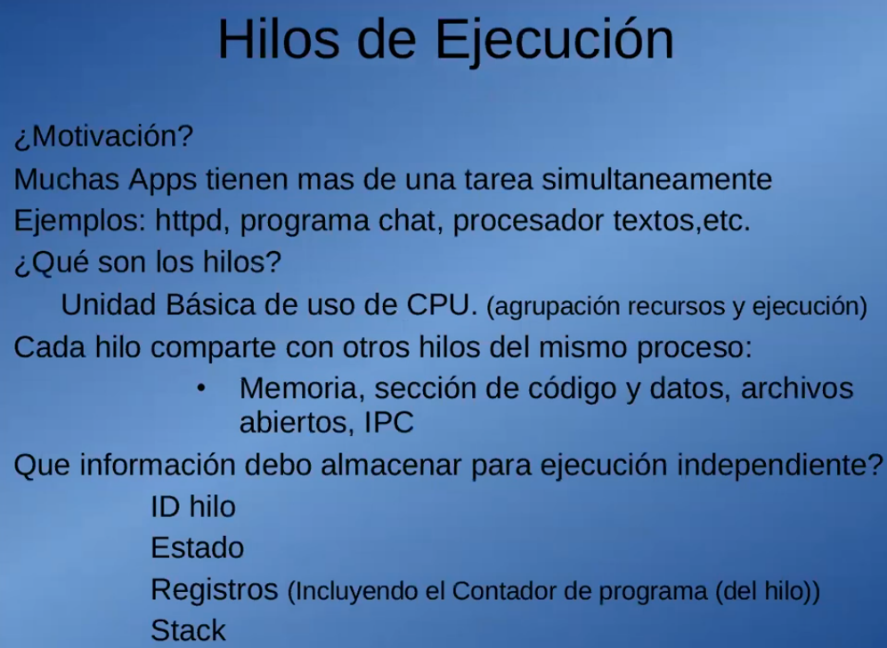
Se bloquearía cuando escribo o no, dependiendo que bloque pongo primero.

Una solución podría ser cada bloque en un proceso distinto y que de alguna manera interactúen.

Otra solución es crear, dentro de un solo proceso, 2 flujos de ejecución. Para hacer esto obviamente, voy a necesitar ayuda del SO. Esta abstracción me sirve para que, por más que haya un hilo bloqueado esperando algo, si hay otro hilo que pueda hacer cosas, se podría ejecutar.

Porque ahora el SO, en vez de planificar o de tener como unidad de ejecución un proceso entero, es como que ahora tiene mas granularidad y ejecuta solo una función/tarea o solo un hilo dentro de un proceso.

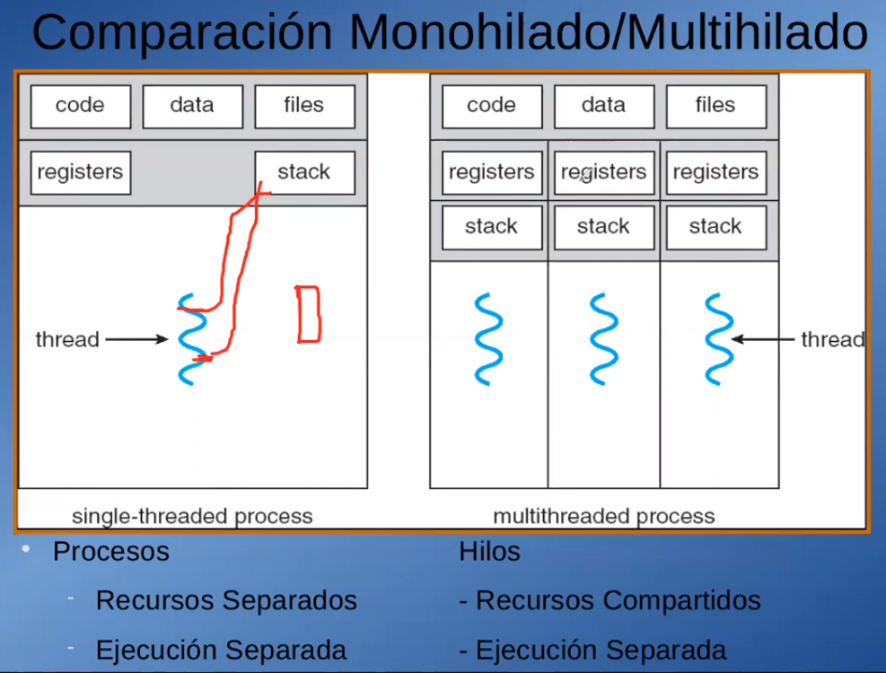
Ejemplo de Editor de texto: Pienso en que escribir (proceso bloqueado), pero cada 3 minutos dice guardando a disco, el corrector ortográfico se está corriendo. Hay varias tareas en simultaneo, puedo decir que lo hago con procesos distintos, pero el tema es que cuando las tareas están muy entrelazadas, supongamos que todas interactúan con todo el documento, yo tendría que usar para los procesos un IPC que comparta todo el documento y avisar cuando uno modifica algo que el otro se entere. Se podría hacer con multi proceso, pero es complejo, es más fácil cuando se implementa con hilos, pero el SO o alguien debe saber de la existencia de que hay mas de un flujo de ejecución dentro de un programa.

La idea es seguir utilizando mas la CPU, minimizar el tiempo que esta ociosa.

¿Porque cree la abstracción de proceso? Para tratar aprovechar los tiempos muertos de un programa en ejecución

¿Para qué creo hilos? Para tratar de aprovechar los tiempos muertos dentro de un proceso cuando se bloquea en una entrada salida. Parte del proceso se bloqueó, pero hay otra parte que puedo ejecutar.

Esto trae aparejado que alguien tiene que saber que hay mas de un flujo de ejecución, y guardar/cargar su contexto o estado cuando se realizan intercambios.



Hay un máximo de hilos por proceso.

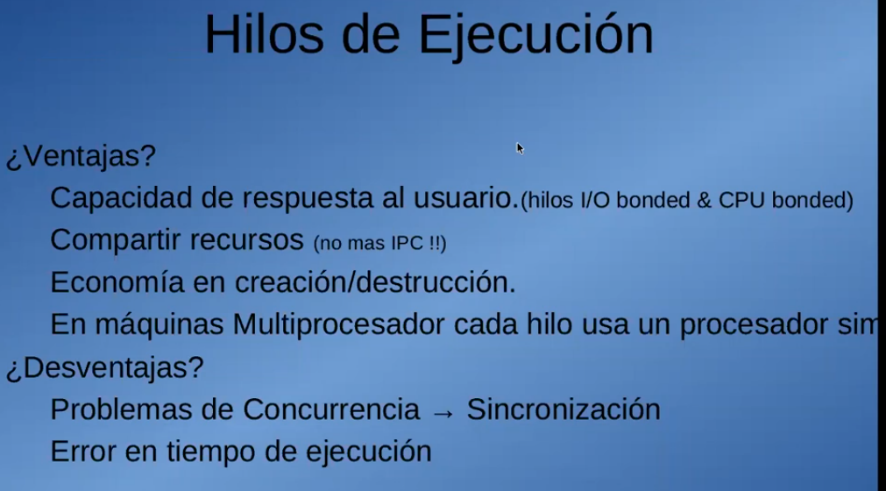
La memoria RAM del proceso es una sola. (yo supongo que habla de la porción asignada a ese proceso)

Cada hilo tiene su pila, para cuando hago push y pop al invocar subrutinas. Para que cada flujo de ejecucion se independiente de los demás, necesito que algunos recursos estén separados.

Ventajas:

No uso IPC

Mas liviano. Ahorro de memoria, ósea viendo la imagen anterior, ante la alternativa de usar 3 procesos distintos. Me ahorro la memoria y los IPC de los 3 procesos.



Realizar un cambio de contexto si tengo soporte de hyper threading o multi hilado es más rápido porque no tengo que remplazar el programa, el código.

- No necesito IPC

- Ocupa menos memoria porque no duplica algunos recursos.

- Es más rápido de crear, y el cambio de contexto.

- En las maquinas multinúcleo, permite ejecutar varios hilos a la vez. Ej: En el caso de la imagen anterior, si tengo 3 núcleos y el proceso 3 hilos, puedo ejecutarlo 3 veces más rápido que si tuviera un núcleo.

Hasta lo que hemos visto ahora para tener psudo paralelismo tenemos 2 alternativas, multi proceso o multi hilo. Obviamente las 2 optimizan el uso de la CPU, pero la comparación de un proceso con uno solo hilo vs uno con varios hilos, es que si separas en 3 procesos es equivalente a 3 hilos pero duplicas los recursos.

Otra ventaja es que la RESPUESTA al usuario es mas rápida. Me refiero a que si antes el proceso era monolítico, y esperaba una entrada de teclado o movimiento del maus, si no lo muevo no hacia nada. Y ahora puedo tener otro hilo que hace otras cosas.

Hasta acá son todas buenas, entonces tiremos multiproceso monolítico a la basura. NO!, hay un par de cosas a tener en cuenta, que van en contra de hilos o le restan puntos.

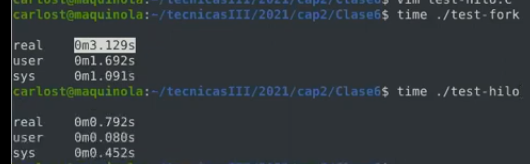
* Un tema son los mecanismos de sincronización. (Los tiene que hacer el programador)

Ósea un hilo A puede acceder a los datos, y otro B puede leer los mismos datos, como hago para que el lector B sepa cuando se terminaron de escribir los datos provenientes de A para poder leerlos.

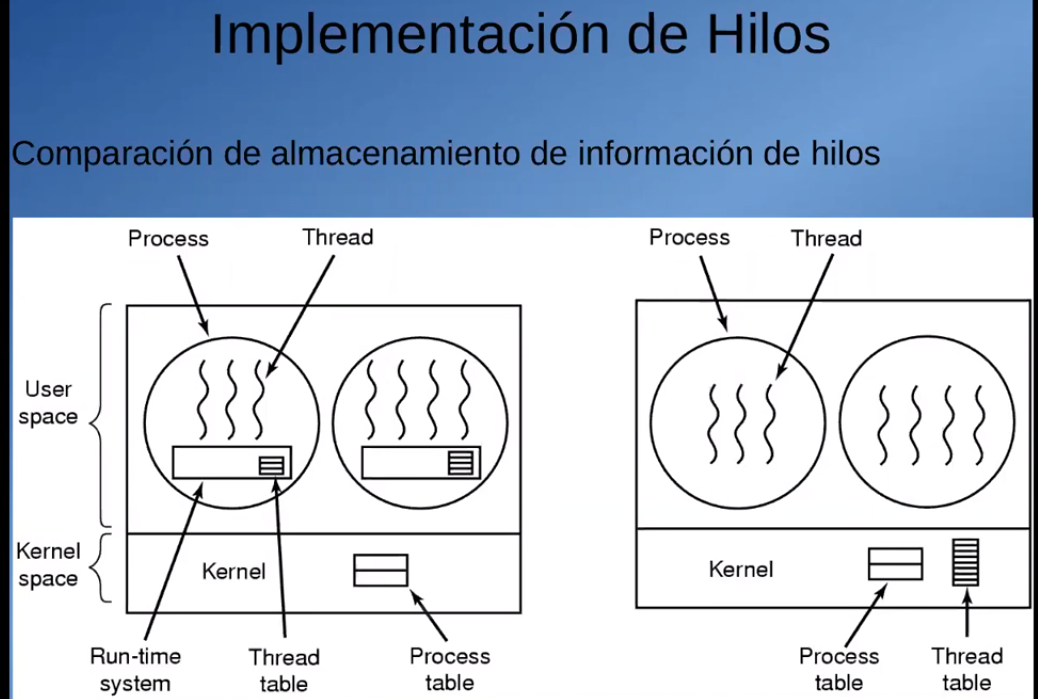
Antes cuando usaba IPC, el SO se encargaba de bloquear el lector hasta que el otro proceso escribía. Con hilos no pasa esto, puede leer y escribir en cualquier momento. No tengo un mecanismo de sincronización, yo programador lo tengo que escribir.

* Otro tema que tiene que ver de como esta implementado. Supongamos que en tiempo de ejecución un hilo da una excepción y termina. Para el SO si cualquiera de los flujos de ejecución de un proceso da un error, termina el proceso, completo. Por lo cual se tiene que ser muy cuidadoso a la hora de programar.

Ejemplo de cuanto demora en crear 10.000 hilos y procesos.



¡Hilos son más rápidos de crear y eliminar!

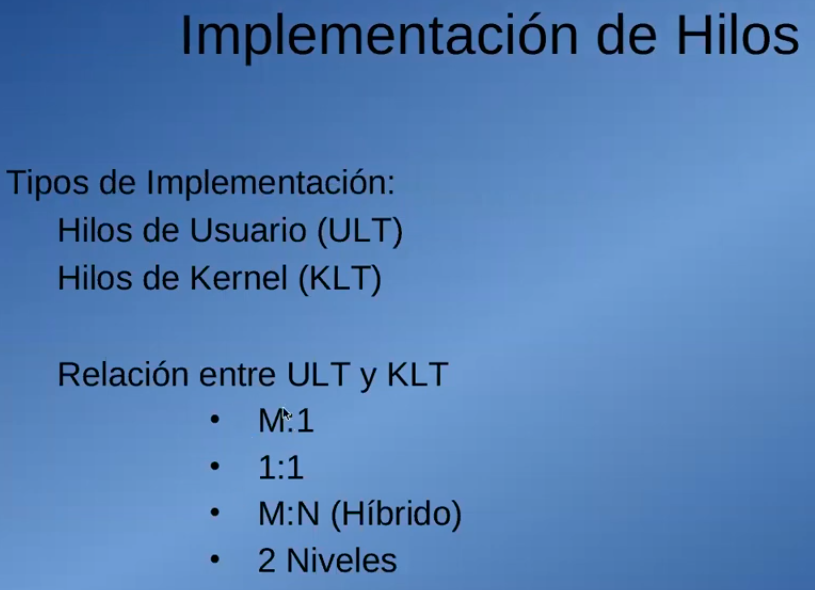
Estos son hilos implementados en kernel, también necesitan llamadas a sistema para crearlos, pero son más rápidas que las de un proceso porque son menos los recursos que hay que asignar. ( Stack y registros especiales)

**¿Cómo los podemos ejecutar o implementar?**

Quien se encarga de crearlos y almacenar la información del hilo.

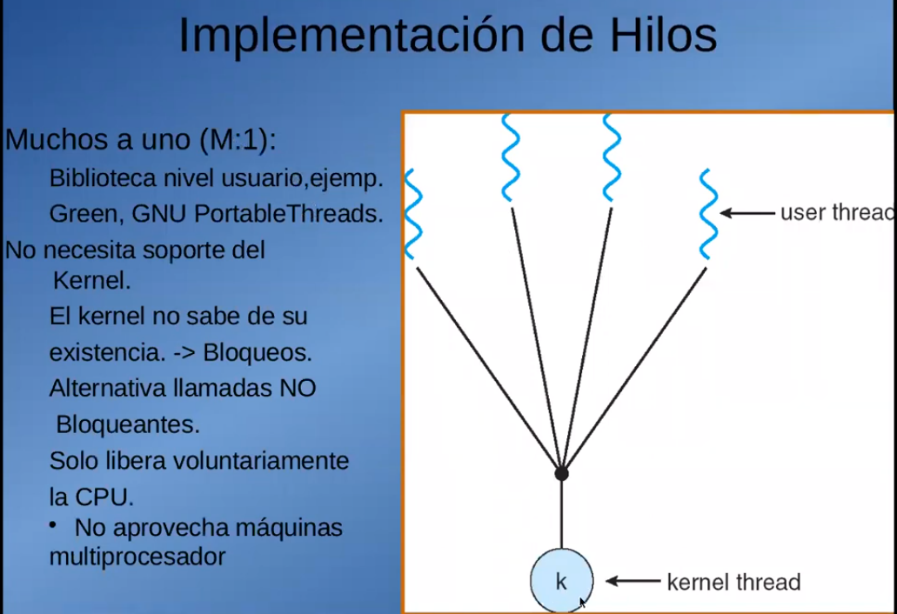
Básicamente hay 2 alternativas. Hilos en usuario o en Kernel.

Podría ser que a nivel usuario se guarde la información del hilo en una biblioteca. Pero a los efectos del SO, solo sabe que hay un proceso, no tiene ni idea que hay 3 hilos.

Otra alternativa es que el Kernel este consciente de que los distintos procesos tienen hilos. Tiene la información de los procesos en la tabla PCB, pero además tiene una tabla de hilos, donde tiene la información de todos los hilos de todos los procesos.

En función de estas alternativas se presentan varios casos de implementaciones.

Múltiples en usuario y 1 en kernel.

M:1 significa que en modo usuario tengo muchos hilos, pero el kernel lo ve como solo un hilo o flujo de ejecución en un proceso.

Ventajas y desventajas:

* En esta implementación el Kernel esta consciente de un hilo mientras que dentro del proceso se guarda la información de los distintos hilos. El tema es que yo necesito un núcleo que soporte hilos, yo puedo tener un SO que no soporte hilos y no puedo modificar el Kernel. Pero en este caso el soporte hilo se da en espacio usuario, necesito correr una aplicación que maneje hilos de usuario. Ejemplos en la imagen

La desventaja de esto, es que si se esta ejecutando algunos de estos hilos, del mismo proceso, y hace una llamada bloqueante al sistema, para el CPU no es el hilo sino todo el proceso lo que esta bloqueado, entonces no podría seguir ejecutándose ninguno de los otros hilos.

Estas bibliotecas lo que hacen normalmente es tener un reaper alrededor de las llamadas al sistema, en vez de hacer una llamada bloqueante tienen una función que se llama “read” o “write”. Entonces lo que hace es revisar si hay algún hilo que puede seguir ejecutándose, entonces conmuta a ese hilo con el objetivo de que no se bloquee el proceso a causa del hilo que va a realizar la llamada al sistema. Si no hubiera ninguno de los hilos que estuviera en condiciones de seguir ejecutándose probablemente esta función si realiza la llamada al sistema bloqueante. Y ahí hasta nuevo aviso no puede avanzar.

Otra alternativa, que pasa si modifico el SO, para que las llamadas al sistema de E/S no fueran bloqueantes, se podría hacer, pero rompe un poco el paradigma que utilizamos y obviamente modificar el SO. Pero ya que lo modificamos al SO, porque no lo hacemos consciente al SO de los hilos, entonces ahí surgieron los SO que tenían implementados hilos en KERNEL.

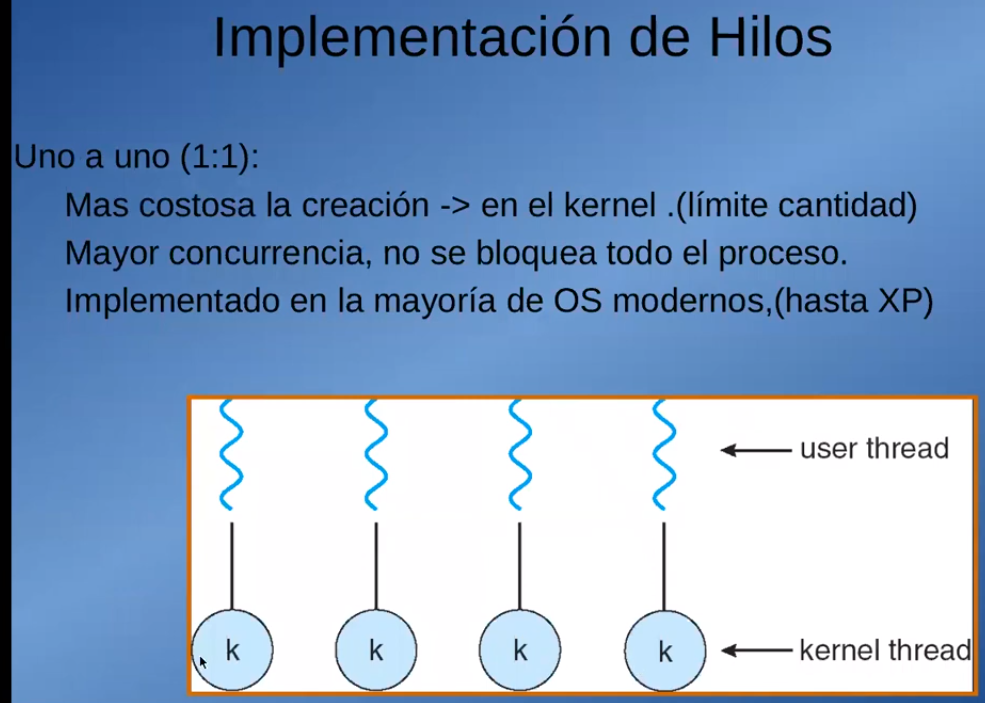
Otra desventaja es que, si tengo varios hilos, podría ser que un hilo se la pasa haciendo cálculos, y ahora no hay un planificador que expropia la CPU después de un tiempo para ser equitativo entre todos los hilos. Los hilos están a nivel usuario, ósea que a lo sumo el planificador me va a sacar todo el proceso. El programador debe realizar por instrucción que los hilos cedan la CPU.

Otra desventaja es que en una maquina multi procesador / multinúcleo no verían los hilos, el SO solo esta consciente de un solo flujo de ejecución por proceso. Y se ejecutaría en un solo núcleo.

* Se puede usar en SO que no soportan hilos a nivel de Kernel.

Contras:

* Si un hilo se bloquea se bloquea todo el proceso. (variante de reapers)
* El programador debe realizar la sesión de CPU para lograr equidad entre los hilos.



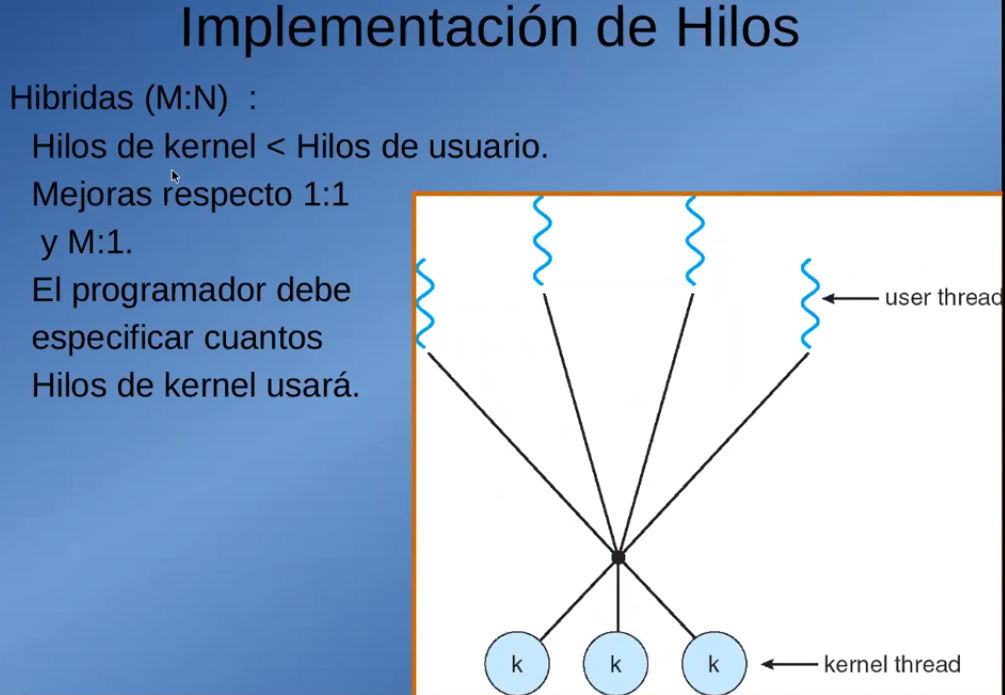
Se dieron cuenta que esto servía y mejoraba el uso de la CPU, entonces lo implementaron en KERNEL y surgió una relación 1:1

-Si se bloquea un hilo no se bloquea el resto

-Es mas caro, en el sentido que cada vez que cree un hilo tengo que pedirle al Kernel que cree la tabla de hilos y le asigne los recursos. No tan caro como crear un nuevo proceso.

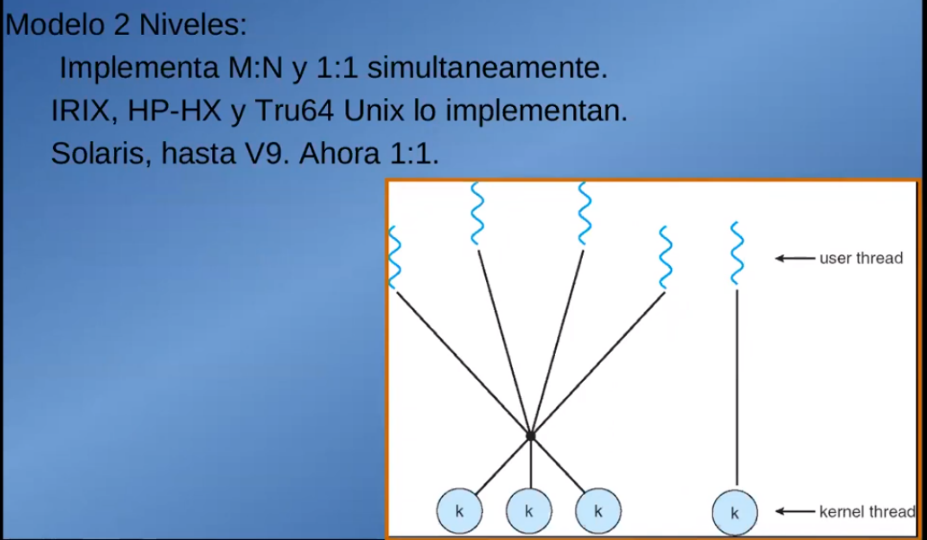
Esta es la mas usada en los SO. Ej: Windows XP.

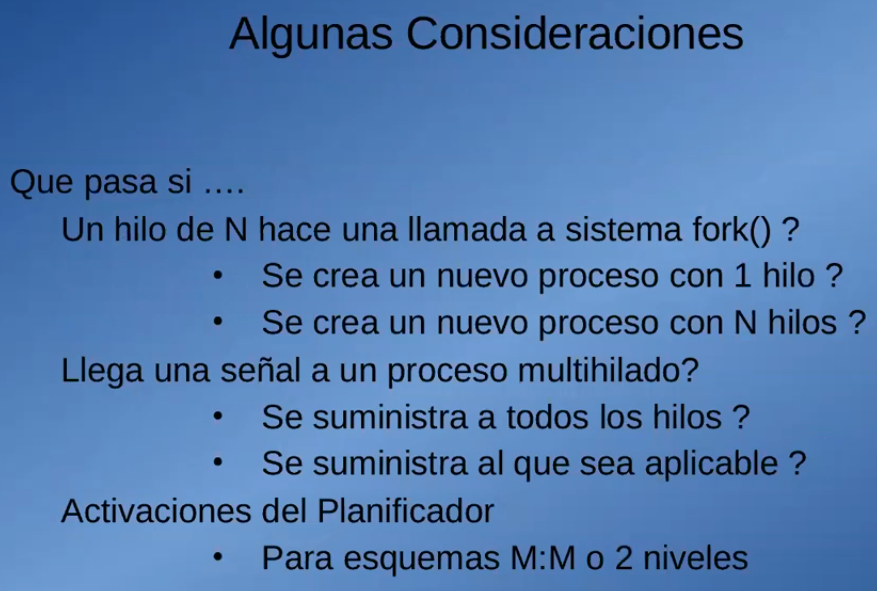
Por simplicidad esta termino siendo esta la mejor aproximación.

Hay un par de otras alternativas que se usaron alguna vez en algunos SO.

Esta trata de mesclar las dos, pero era complicado.

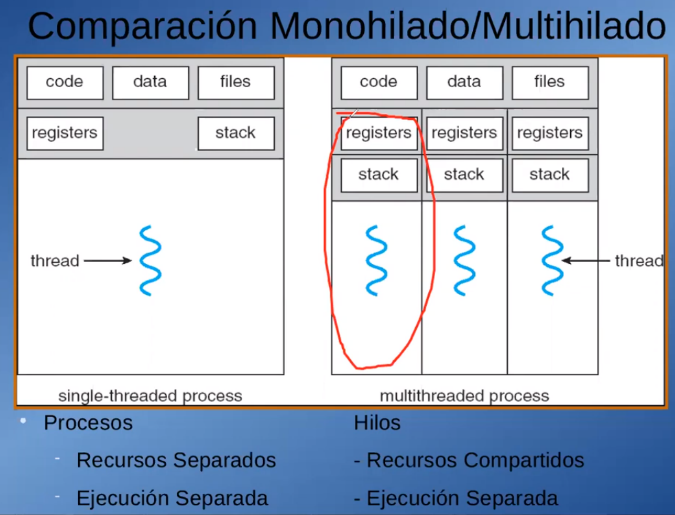
La imagen no me convence. La idea era por ejemplo para 10 hilos de usuario podía crear 4 de Kernel. Entonces 1 de kernel manejaba 2 o 3 de usuario.

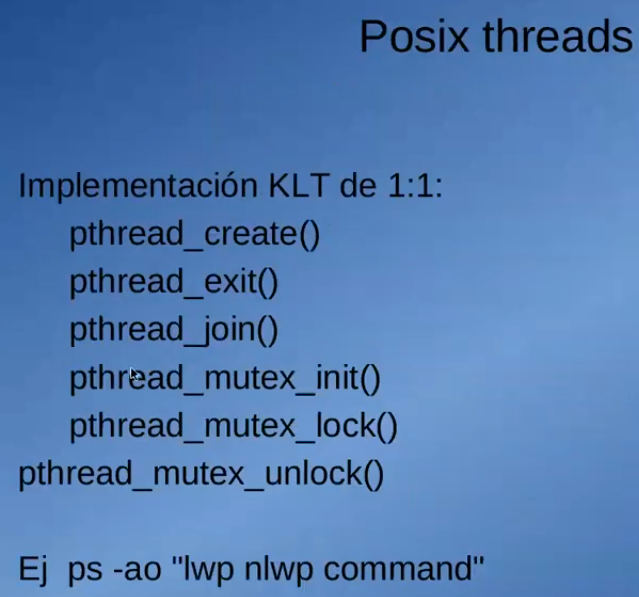
Esta soportaba ambas M:N y 1:1. Pero también complicado, como defino que hilo de Kenerl maneja que hilos de usuario. La asignación era compleja.

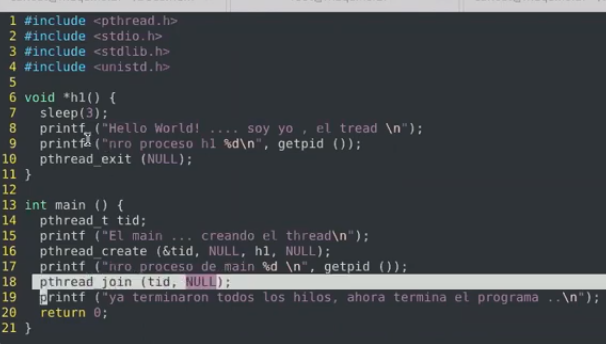


Todos estos temas dependen de que SO se este utilizando. Cada SO lo definió a su manera a su criterio. La idea no es hacer este tipo de cosas porque el resultado puede ser inesperado.

Ósea si usamos hilos no usemos fork() ni señales. Por las dudas.

Un hilo es uno de los “n” posibles flujos de ejecución con su contexto, dentro de un proceso, pero no un mini proceso. NO tiene código, por si mismo no ejecuta nada. NO tiene razón de ser si no esta dentro del contexto de un proceso.

En Linux se implementa 1:1, y tengo que hacer llamadas al sistema y lo puedo crear.

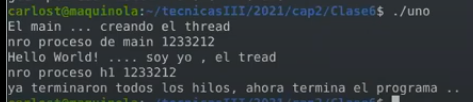
****

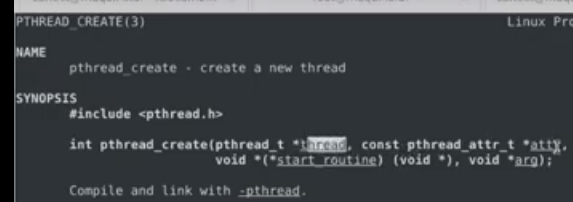
**Ejemplos:**

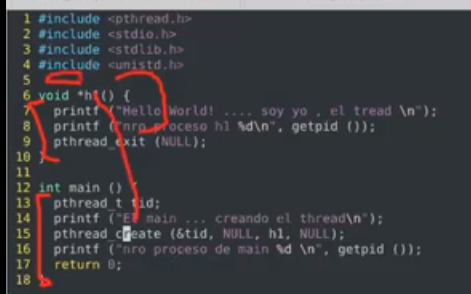
Lo mas interesante y distinto de fork(); es que, primero no crea un proceso, y segundo, que yo le digo que flujo de ejecución ejecutar.

Le digo ejecuta esta función.

Es más elegante decir, mi tarea la separo en varias partes, creo funciones. Y a cada función la invoco con un hilo. Y cada uno realiza esa función y nada más.

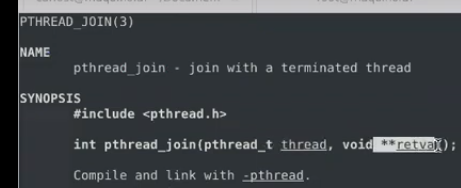
Los argumentos de pthread\_create(1,2,3,4):

1. un ID, que no lo pongo, sino que me lo devuelve cuando retorna el hilo. Para identificarlo. Un puntero a una variable tipo pthread\_t.
2. Una estructura de atributos. Un puntero. Puedo modificar el tamaño del stack del hilo, y podría modificar el algoritmo de planificar el hilo (prioridades)
3. La función que ejecuta
4. Los argumentos de la función que va a ejecutar.

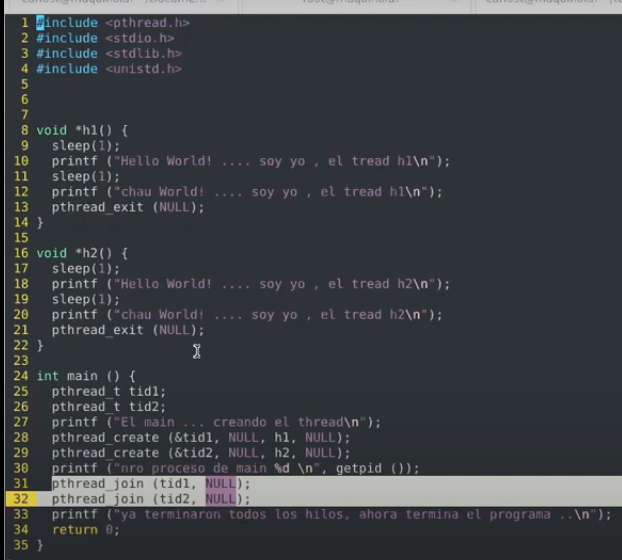
Explica el uso de variables globales y locales del main y de hilo. Solo comparten la global.

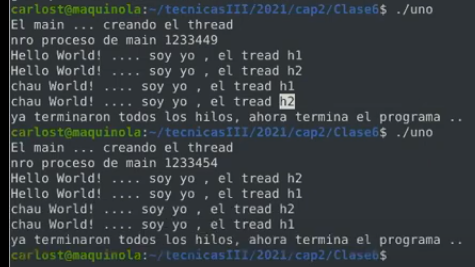
Sin el sleep daba error. Porque se crea el hilo, pero el hilo main terminaba y cerraba todo el proceso.

Si pongo pthread\_exit(NULL) en el hilo main, espera a que terminen los demás hilos.

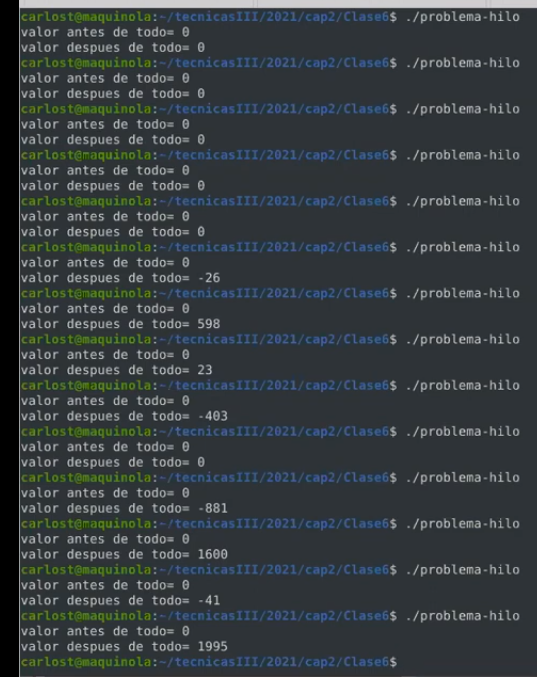


Con el exit pudo mandar un código y recuperarlo con el join





EL planificador los ejecuta como se le canta. No siempre se ejecutan de la misma manera.



¿Qué paso acá?

Tiene problemas de concurrencia. Hay 2 entidades que quieren hacer algo a la vez. Las instrucciones no son atómicas.

Esto no es determinístico. Cuando lo corro no siempre da el mismo resultado.

