|  |
| --- |
|  |
| UD 2: Programación multiproceso (I) |
|  |
| Programación de servicios y procesos |



**Centro de estudios SEIM**

4 de septiembre de 2012

Autor: Daniel Miguel

UD 2: Programación multiproceso (I)

Programación de servicios y procesos

Proceso

En términos generales se puede decir que un proceso es un programa que se ejecuta en el sistema operativo.

Más en profundidad, podríamos decir que es una actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual y un conjunto de recursos del sistema asociados. En otras palabras, el sistema operativo lanza un proceso para la ejecución de un programa.

Cada proceso tiene su contador de programa (puntero de instrucciones, registro que contiene la dirección de la instrucción que se está ejecutando en cada momento), registros y variables de memoria aislados del resto de sus programas, y tiempo de CPU asignado.

Cuando dos procesos pertenecen al mismo programa ejecutado 2 veces, cosa que es perfectamente posible, el sistema operativo también aísla los registros y variables de memoria utilizados por ellos, aunque comparten el mismo segmento de código, debido a que ésta no cambiará.

De forma más general un proceso consta de:

* Uno o más hilos (unidades de ejecución).
* El estado de ejecución, esto es, los valores de los registros de la CPU para dicho programa.
* Su memoria de trabajo, es decir, la que tiene reservada y su contenido. Se dice que es su memoria crítica, porque ningún otro proceso tiene acceso a la misma.
* Información para la planificación. El planificador de procesos se encarga de repartir el tiempo de CPU entre los distintos procesos en ejecución. Lo habitual es que el reparto se haga en base a una cola circular (Round Robin) aunque también existen otros métodos como FIFO o LIFO. Es posible que la planificación utilice un sistema de prioridades a la hora de repartir el tiempo.

Los procesos son lanzados por el S.O, quien también se encarga de la comunicación entre procesos a petición, esos sí, de otros procesos (aplicaciones multiproceso).

En los sistemas multihilo, un proceso puede lanzar varias líneas de ejecución paralelas (hilos) que compartirán el mismo espacio de memoria, aunque cada uno guarde su propio estado (registros del sistema y variables de memoria).

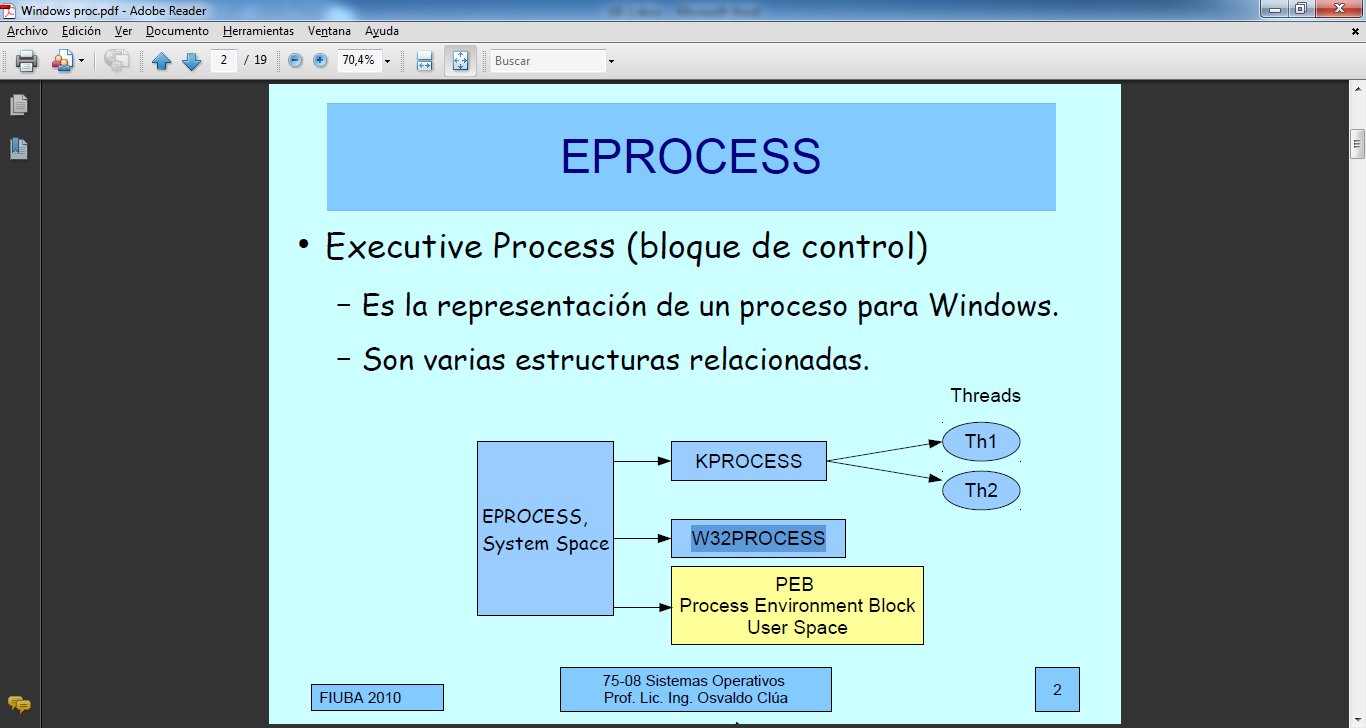
La comunicación entre procesos es más costosa en términos de recursos que la comunicación entre hilos porque los primeros no comparten el mismo espacio de memoria y además, el sistema aísla el espacio de memoria de los procesos.

Crear un nuevo proceso, también es más costoso que lanzar varios hilos desde un mismo proceso por motivos similares a los que se han explicado.

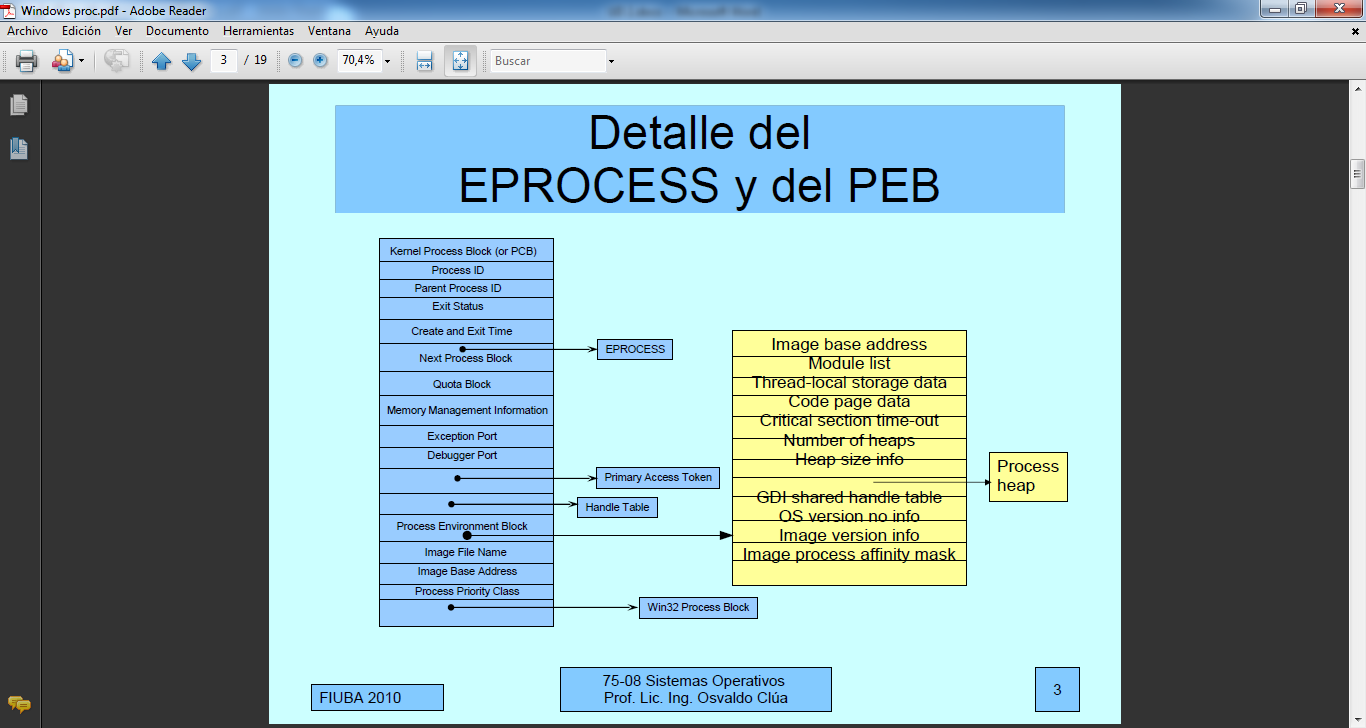
Los procesos en Windows

<https://sites.google.com/site/sarvasite/windows-os-internals/processes-threads/process-internals>

Windows representa los procesos mediante la estructura EPROCESS. En realidad son varias estructuras relacionadas como vemos en la siguiente figura.



El desglose de las estructuras EPROCESS y PEB es el siguiente:



Mecanismo de ejecución de un proceso

# https://sites.google.com/site/sarvasite/windows-os-internals/processes-threads/process-internals/ProcessCreationStages.JPG?attredirects=0

# Abrir el archivo imagen.

En primer lugar se define la prioridad con la que se va a crear el proceso. A continuación, se asocia el proceso a un Escritorio y se verifica si el usuario tiene permiso para ejecutar esa imagen según las políticas actuales de seguridad.

Se asocia el proceso con un subsistema (Win 16, Win32,WoW o Dos). Si la imagen no es la nativa, se invoca la imagen correspondiente.

Por último si hay un intérprete predefinido en el Registro (Debugger o Just In Time), se invoca.

# Crear el proceso.

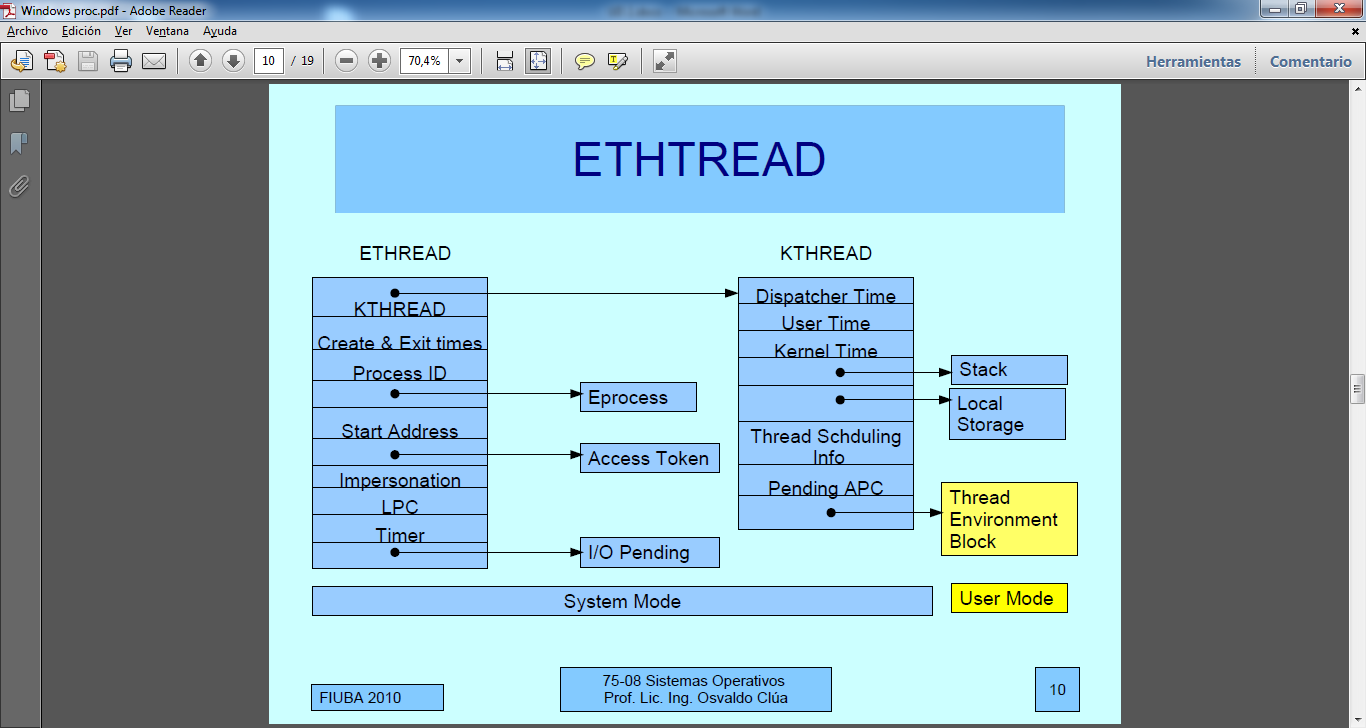
Primero prepara el bloque EPROCESS. Después se prepara el espacio inicial de direcciones de memoria donde se alojará el proceso.

El siguiente paso es crear el bloque KPROCESS y mapear los recursos del executive (ntdll,dl, Nat. Language, etc) al espacio de direcciones.

Por último preparará el PEB y encadenará el EPROCESS aunque todavía no se puede ejecutar hasta que no iniciemos un Thread, que es la unidad de ejecución.

# Crear el Thread inicial

Lo primero que se hace es incrementar el Thread Count del proceso y preparar la estructura ETHREAD para el Thread inicial. En este paso le agrega un id.



Prepara el Bloque Thread Environment Block de memoria y mapea la dirección de comienzo. Inicializa las propiedades que dependen del sistema como la prioridad o la afinidad (características que determinan el procesador en el que preferentemente se ejecutará el thread).

# Notificación al subsistema.

Crea las estructuras para que el CSRSS lo maneje, muestra el cursor de reloj de arena durante 7 segundos máximo y si el proceso no muestra ventana, vuelve al cursor original.

# Pasos finales.

Inicializa el heap y demás estructuras dinámicas. Carga las .dll (invocándolas directamente si son en código nativo, o interpretando el assembly)

Por último prepara un (APC) para ejecutar el proceso. Se trata de un procedimiento de llamada asíncrona, que interrumpe la ejecución de un hilo para lanzar el nuevo proceso.