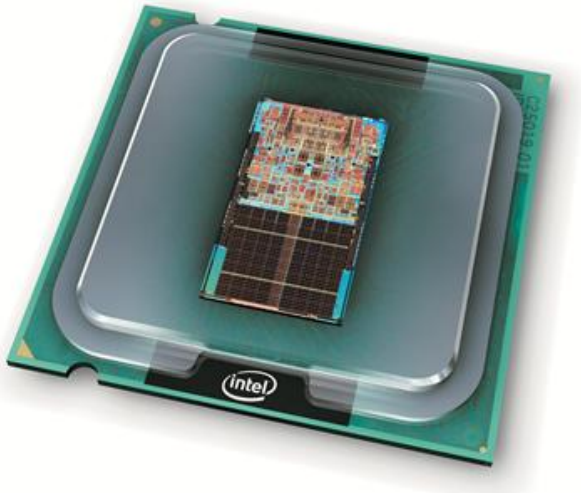



# Procesadores IA-32 e Intel® 64
# Inicialización

Alejandro Furfaro

Abril 2013

# Agenda

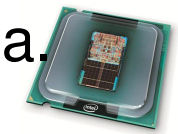
1. Inicialización de un computador
   - ¿Como arranca el procesador?
   - Análisis Conceptual de la tarea
2. Modo Protegido
   - Responde a un requerimiento
   - Vamos a trabajar en Modo Protegido
3. Arrancando en 64 bits
   - Una píldora roja mas... “ancha”

# ... erase una vez... el Modo Real

## Algunos valores de arranque de un procesador IA-32

- **EIP** = 0x0000FFF0
- **CS** = 0xF000 *<-Esta es la parte visible*. Como veremos, **CS** tiene una parte oculta, que luego del reset toma los siguientes valores:
  - **Base** = 0xFFFF0000
  - **Límite** = 0x0FFFF
  - **Atrib** = P=1:DPL=xx:S=1:Tipo=101:A=1
- **Base** + **EIP** = 0xFFFF0000 + 0x0000FFF0 = 0xFFFFFFF0

- A pesar de estar en Modo Real, busca su primer instrucción mucho mas allá del 1er. Mbyte.
- A solo 16 bytes del fondo del espacio de 4 Gbytes, la alternativa es saltar hacia alguna dirección de memoria que permita trabajar mas cómodo.
- Ni bien cambia el registro **CS**, se desactiva la posibilidad de generar direcciones por encima del primer Mbyte de memoria.
- Debe saltar hacia una dirección dentro del 1er. Mbyte.



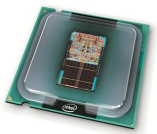
# Pasos siguientes

## Organización mínima

- El sistema debe tener una memoria no volátil en el espacio que contenga la dirección 0xFFFFFFF0. Es la única forma de poder empezar.
- Sin embargo debe haber alguna memoria de igual tecnología en el espacio de direccionamiento al que se salta desde 0xFFFFFFF0, ya que de lo contrario en sistema no puede proseguir. Necesitamos en ese lugar de la memoria,una cantidad mínima de código que ponga en operación a nuestro computador. Esto supone:
  - Inicializar el hardware básico que permita manejar el refresco de la memoria dinámica.
  - Inicializar los principales dispositivos de E/S que prestan el primer nivel de soporte al procesador.
  - Inicializar el sistema de vectorización de Interrupciones.
  - Poner en funcionamiento al software que administrará el computador: Un Sistema Operativo. Mas simple o mas complejo.

# ¿Quien dará los pasos siguientes?

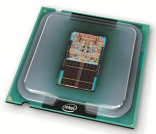
- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático SO que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó el SO que usas?. Pensá.....



# Multiple choice. Necesario aprobar para seguir adelante

¿Cual es la skill de la gente que diseñó tu SO?.

- **a.** Bombero
- **b.** Odontólogo
- **c.** Fisioterapeuta
- **d.** Chef
- **e.** Licenciado en Ciencias de la Computación
- **f.** Vendedor de Seguros
- **g.** Periodista
- **h.** Abogado
- **i.** Actor



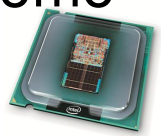
# Solo si aprobaste el Multiple choice. Sino... Loop prev_slide

## La cuestión es dejar de pensar y actuar como usuarios

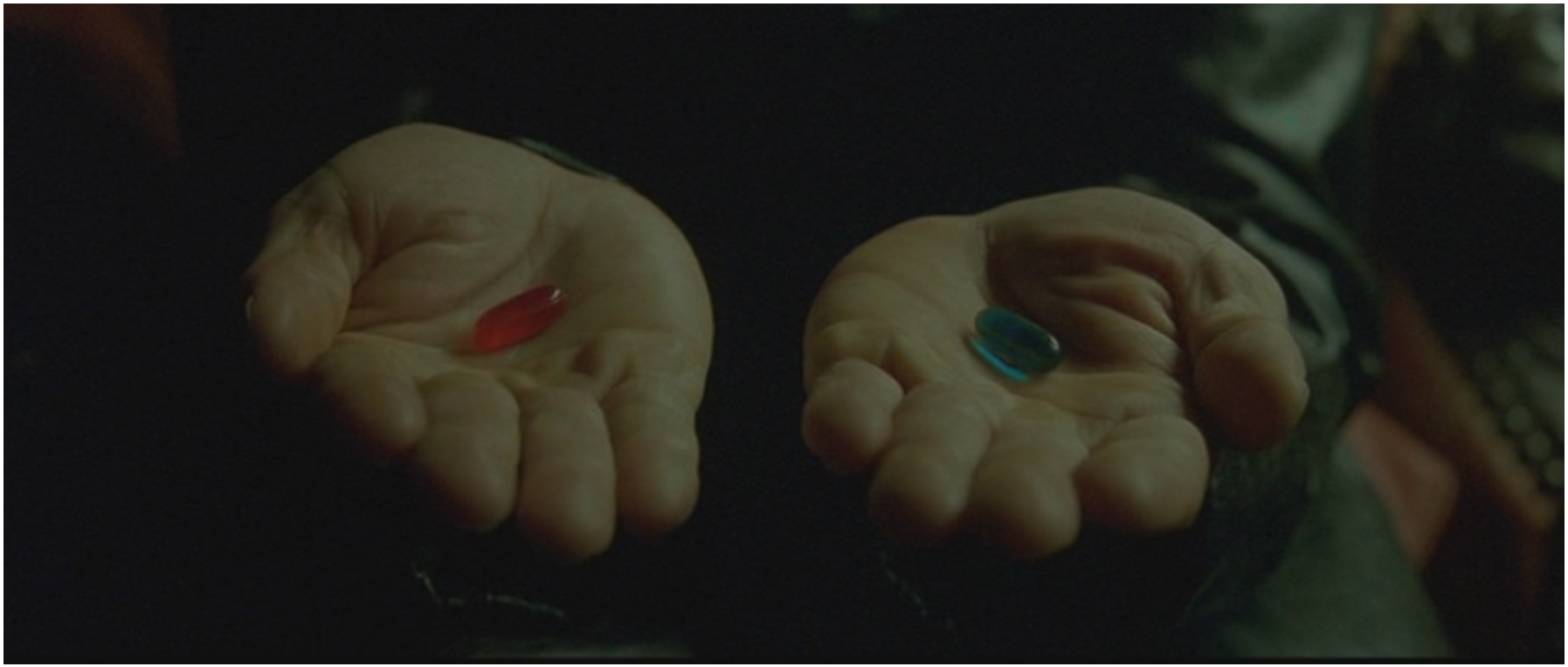
- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la E/S. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.

# Solo para Expertos en Ciencias de la Computación

- Lo dicho en el slide anterior no es válido para usuarios generales, como por ejemplo el resto de las opciones del múltiple choice anterior. Mas aún. Para este público el comportamiento descripto es el esperable.
- Pero Uds. son **otro público**, básicamente porque **eligieron serlo**. Por eso están aquí. ¿no?
- Así que a enterarse: Tu trabajo es y será siempre, **entender** como funcionan los sistemas y tecnologías de informática que te toque enfrentar, para poderlos diseñar, mejorar, o corregir. En este caso, la arquitectura de un computador, pero lo mismo vale para un algoritmo, o un sistema de inteligencia artificial.
- Creeme: Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar la tecnología. Algo que pesando como usuario no vas a conseguir...



# Es momento de revalidar la elección



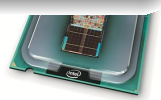
## The choice

O nos quedamos con el wizard que nos resuelve la vida sin tener que pensar... O nos decidimos a enfrentar las cosas como son realmente, y entenderlas, aprendiendo, si es necesario, a hacer todo desde cero y a pulmón.

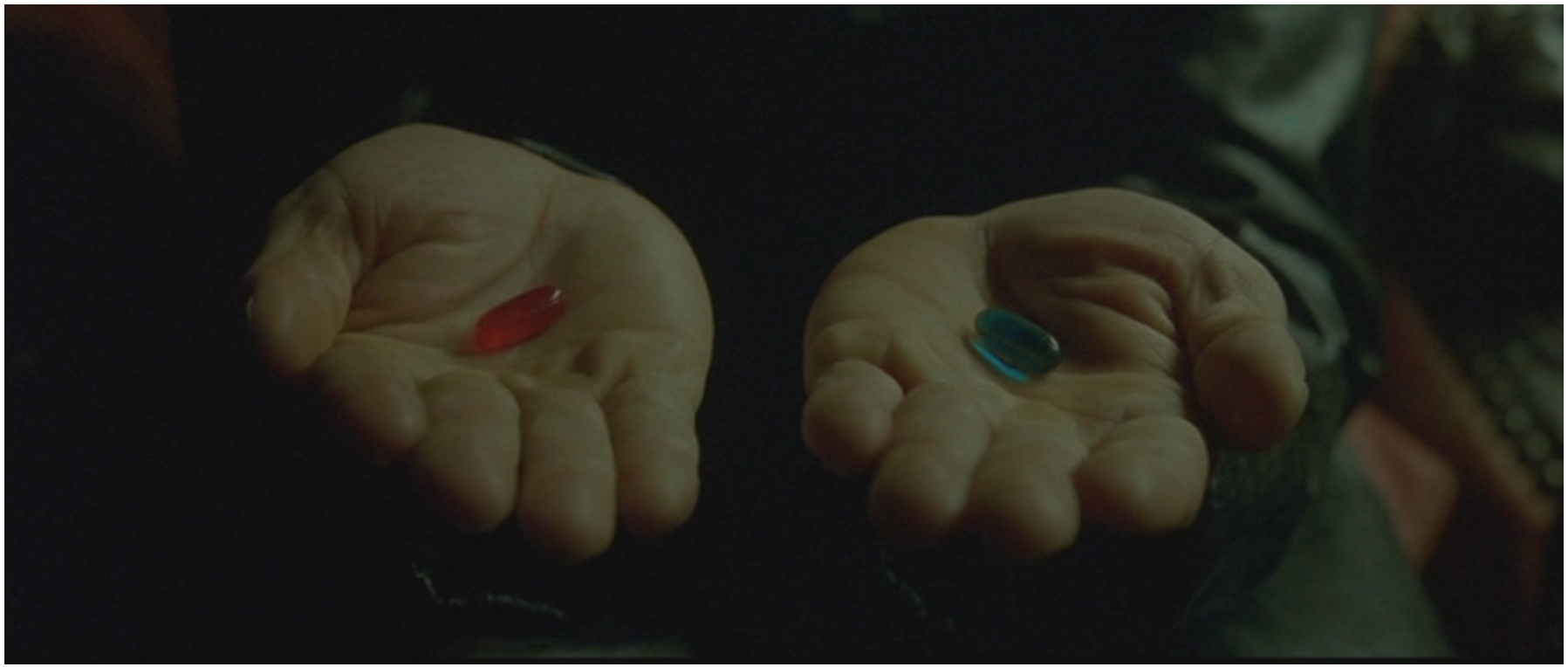
# Retomando nuestro problema

## Arrancamos el procesador y estamos en modo real

Para iniciar su operación en modo real no se requiere mas que inicializar un sistema de interrupciones con las reglas de modo real (para un 8086 diseñado en 1978), con los vectores de interrupción apuntando a código diseñado para operar en Modo Real, y tener disponibles las correspondientes funciones para cada handler de interrupción.
Si nos pensamos quedar en este pequeño microclima de confort (desperdiciando el 99,999 % de los recursos del procesador :-/ ), se necesita un mínimo kernel que administre la ejecución de los programas que compongan en rango de aplicaciones, ejecutándolas una a la vez (leíste bien... Primero una , y recién cuando finaliza, podés ejecutar otra aplicación).



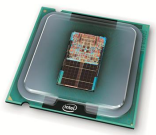
# Otra vez...



## The choice

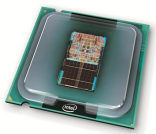
La píldora azul nos deja en este nanoclima confortable. Fin de la presentación.
La píldora roja nos lleva al mundo real, donde utilizaremos TODOS los recursos del procesador para construir un Sistema Operativo...aunque no nos guste lo que vamos a encontrar.

# FAQ's



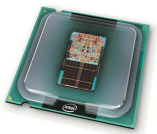
# Q: ¿Vamos a desarrollar un sistema operativo???

A: La construcción de un Sistema Operativo es a veces una tarea gigantesca, como por ejemplo Linux, pero en ocasiones puede requerir un número mucho menor de rutinas que, aunque de muy bajo nivel, provean un conjunto de recursos base suficientes para administrar un sistema de menor tamaño, como un embeeded system.



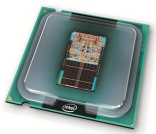
# Q: ¿Para que?¿O acaso un embedded system no puede hostear Linux?

A: Por supuesto. $\mu$CLinux por ejemplo es una implementación de Linux para Embedded. Las embedded PC basadas en procesadores Atom, tambien pueden aceptar un Linux cualquiera. Aunque otros embedded no tienen un procesador con recursos de hardware suficiente para soportar Linux.

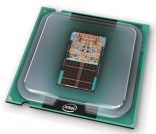


# Q: ¿Para que necesitamos saber esto entonces? ¿No está todo hecho?

A: ¿Otra vez pensando como usuario? Además si un tal Linus Torvalds se hubiese quedado en esta pregunta hoy solo existiría Windows como alternativa para nuestra PC. (Vade retro!)

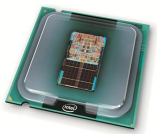


# Ingresando a modo protegido



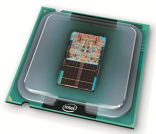
# Requerimientos de los Sistemas Operativos Multitasking

- Área de memoria exclusiva para cada tarea para almacenar su código y sus datos. (Área Local).
- Área de memoria común a todas las aplicaciones, para que éstas puedan acceder a datos globales del sistema, o a código propio del Sistema Operativo de modo de permitir la comunicación entre las aplicaciones. (Área Global).
- Cada tarea podrá acceder únicamente a su Área Local y al Área Global, pero nunca podrá acceder al Área Local de otra tarea. De este modo el Sistema Operativo garantiza la integridad (PROTECCION ;) ) del código y de los datos propios de cada tarea.
- Alta velocidad de procesamiento
- Gran capacidad de Direccionamiento de memoria



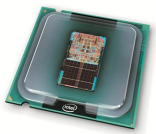
# Requerimientos de los Sistemas Operativos Multitasking

- Amplio espacio de direccionamiento para memoria RAM
- Capacidad de Gestión de memoria de cada tarea por el método de Memoria Virtual
- Capacidad de implementar Multitarea de manera rápida y segura.
- En cada momento la CPU ejecuta una tarea de la lista que mantiene, poniendo a su disposición todos los recursos de hardware de la máquina, incluyendo la cantidad de memoria requerida por la aplicación.



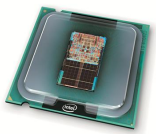
# Finalmente... ¿Que es Modo Protegido?

- Es el conjunto de recursos de hardware y sus reglas de funcionamiento que se requieren para darle sustento a un sistema operativo multitasking satisfaciendo los requerimientos anteriores.
- Su dominio permite entender como funcionan las cosas en un sistema real que puede ser un super servidor, o un embedded.Y en definitiva es nuestro trabajo.
- Es decir,... es tomar la píldora roja.



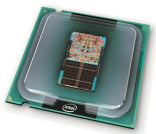
# ¿Como encarar la tarea?

Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.



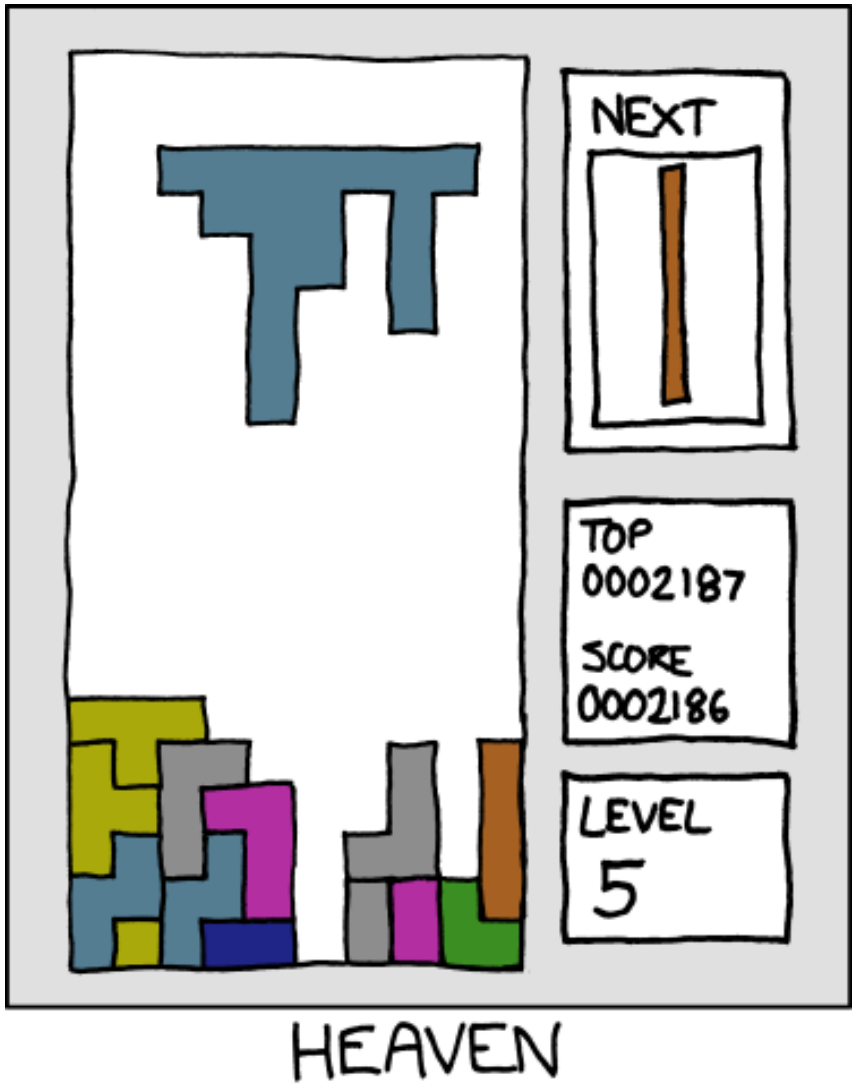
# ¿Como encarar la tarea?

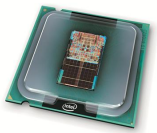
Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.



# No vamos a mentir. No es fácil

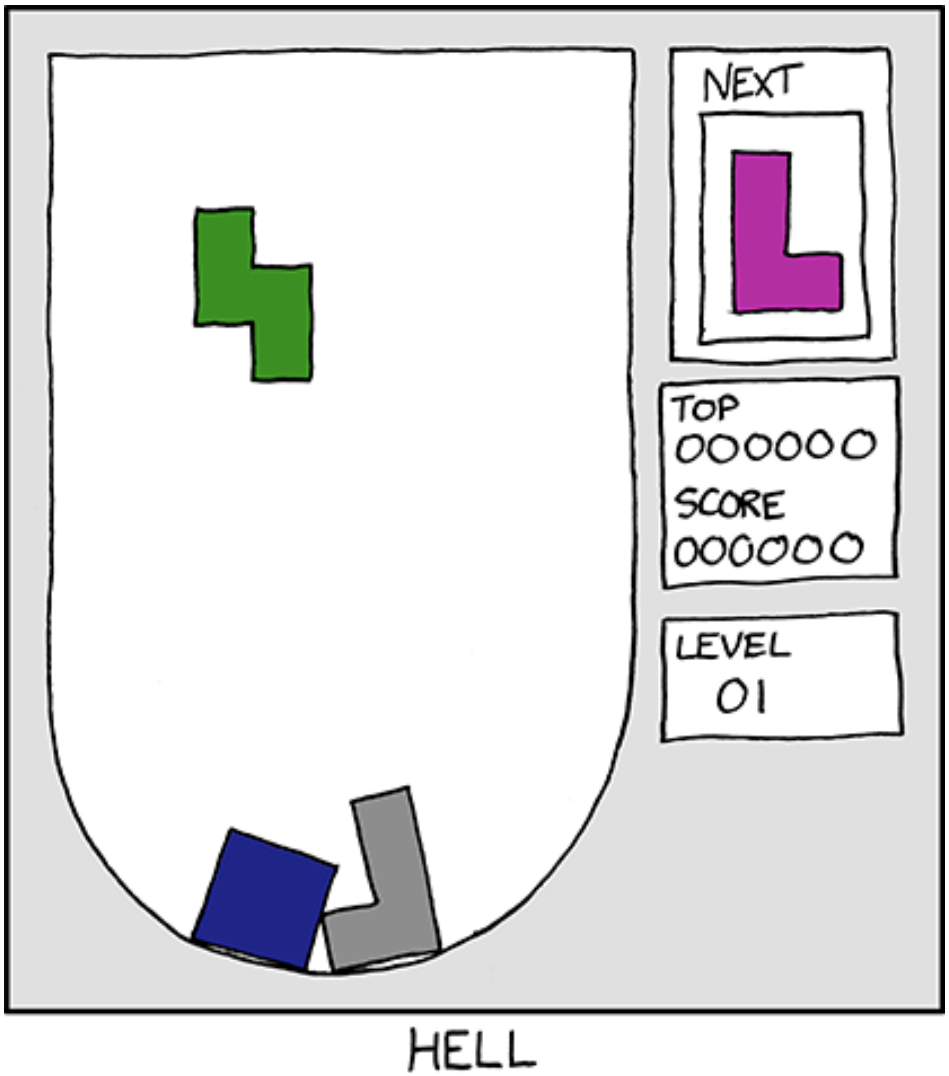
Programación a nivel de aplicación...

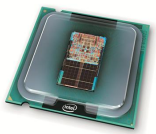


# En ocasiones vas a experimentar algo como...

Programación en Modo Protegido desde cero...



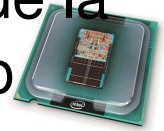



# “Wellcome... To the real world”

**¡Importante!**

Para pasar a Modo Protegido, es suficiente con setear el bit ***CRO.PE***. Pero antes de esta simple e inocente acción, es necesario preparar un entorno mínimo para que todo funcione adecuadamente, y no se estrelle el nano kernel que nos proponemos desarrollar.

1. Programar una ***IDT*** con los descriptores de Excepciones necesarios para operar el sistema, cada uno referenciado a un mínimo handler que al menos sirva para detener el procesador en caso de generarse alguna excepción, y con los descriptores de Interrupción apuntados a los handlers de interrupción del hardware que necesitamos utilizar.
2. Programar una ***GDT*** con los descriptores de Código y Datos que necesitemos mínimamente para iniciar el sistema.
3. Inicializar el registro ***GDTR***, con la dirección base y tamaño de la tabla respectiva. ***IDTR*** puede inicializarse aquí, o ya en modo protegido pero **siempre antes de habilitar las interrupciones**.



# “Wellcome... To the real world”

4. Armar al menos una ***TSS***.
5. Armar una ***LDT*** (opcional... solo para valientes :–) )
6. Si vamos a administrar la memoria por paginación, entonces hay que armar al menos un Directorio de Tabla de Páginas, con al menos una entrada válida que referencia a una Tabla de Páginas, que a su vez debe estar correctamente inicializada con las referencias a las áreas de memoria que vamos a utilizar al inicio de la operación del sistema.
7. Un segmento de Código que contenga el código que vamos a ejecutar ni bien se ponga al procesador en Modo Protegido, y los handlers de interrupción y de excepción. Este segmento debe tener un descriptor debidamente inicializado en la ***GDT***.
8. Inicializar los registros de Control del procesador. ***CR3*** con la dirección física de inicio del Directorio de Tablas de Páginas, y en ***CR4***, setear algún modo de paginación larga en caso de querer utilizarse (Hacerlo en modo protegido con la Paginación habilitada genera una excepción #GP).

# ¿Y ahora?

## Hay que armar paulatinamente un kernel

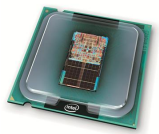
Las definiciones que se tomen dependen de cada proyecto.
Se debe definir un scheduler que se invoque periódicamente (desde el timer tick típicamente), capaz de de conmutar entre las diferentes tareas, asignándoles prioridades, y demás.
Diseñar un sistema de protección que asegure que cada tarea tenga un área de memoria exclusiva inaccesible por el resto de las tareas del sistema, acceso a los servicios del kernel, trabajando las aplicaciones en Modo User y el Kernel en el máximo nivel de privilegio.
Un sistema de Device Drivers consistente, capaz de manejar una consola, un disco, un mouse, un dispositivo de comunicación serie, etc. Un file system para alojar las tareas en forma de archivos.
Un loader para acceder al disco y cargar las tareas en memoria para su ejecución.
etc. etc

# Para trabajar en 64 bits hay que estar en Modo Protegido

**¡Importante!**

Para pasar a modo IA-32e es necesario partir de Modo Protegido, y tener previamente el bit ***CR4.PAE*** activo para (modo largo).
Asegurarse que todo el código que realizará la operación descripta a continuación resida en una página con identity mapping.

1. Si el procesador está en modo protegido con paginación habilitada y no está habilitado Physical Address Extension (PAE), antes que nada poner ***CR0.PG*** en 0 para deshabilitar paginación.
2. Habilitar Physical Address Extension, poniendo ***CR4.PAE*** = 1.
3. Inicializar ***CR3*** con la dirección física en que inicia el Page Map Directory Level 4 (PLM4).
4. Habilitar el Modo IA-32e, seteando el bit LME del Model Specific Register ***IA32_EFER***.
5. Habilitar paginación con ***CR0.PG*** = 1. Esto provocará que el procesador setee además el bit ***IA32_EFER.LMA***.



# ¿Nada mas?

Las Tablas de Paginación deben residir en los primeros 4 Gbytes de memoria antes de habilitar el modo IA-32e.

## *Condiciones*

El procesador chequea los bits ***CR0.PG***, ***CR4.PAE***, y ***IA32_EFER.LME*** para detectar inconsistencias. En cualquiera de los casos siguientes genera una excepción #GP.

- Si se está en modo protegido y se intenta entrar o salir del modo IA-32e con ***CR0.PG*** = 1
- Si en modo IA-32e se activa PAE antes de activar Paginación.
- Si se desactiva PAE en Modo IA-32e
- Si se intenta activar el modo IA-32e desde un código ubicado en un segmento de código de 64 bits.
- Si el registro TR tiene cargada una TSS de 16 bits.

# Arrancando directo a 64 bits

Si nuestra idea es habilitar el modo IA-32e directamente, algunas cosas se pueden hacer en Modo Real.

1. En modo real armar una ***GDT*** con al menos un segmento de 64 bit de código y otro de datos.
2. Inicializar el registro ***GDTR*** con los valores de dirección base y límite de la ***GDT***.
3. Activar PAE. Aun en modo Real podemos hacerlo, usando MOV a ***CR4***, siempre antes de activar Paginación.
4. Setear el bit LME del Model Specific Register ***IA32_EFER***. En Modo Real no tiene efecto pero al estar también PAE activado, ni bien entremos a Modo Protegido se activará directamente el modo IA-32e.
5. En ***CR0***, activar al mismo tiempo la paginación y el bit PE.
6. En este punto estamos en Modo Compatibilidad de 16 bits (ya que en la instrucción anterior estábamos en Modo Real), de modo que solo resta un salto far al segmento de 64 bits definido en la ***GDT***