

# Práctica de Organización del Computador II

System Programming

---

Segundo Cuatrimestre 2023

Organización del Computador II  
DC - UBA

# Introducción

---

En la clase de hoy vamos a ver:

En la clase de hoy vamos a ver:

- **Nociones de System Programming**

En la clase de hoy vamos a ver:

- **Nociones de System Programming**
- **Modo Real y Modo Protegido**

En la clase de hoy vamos a ver:

- **Nociones de System Programming**
- **Modo Real y Modo Protegido**
- **Pasaje a modo protegido**

# System Programming

---

¿Qué es System Programming?



¿Qué es System Programming?

- Su objetivo es producir software que genere abstracciones útiles y eficientes en base a los recursos de hardware disponibles

## ¿Qué es System Programming?

- Su objetivo es producir software que genere abstracciones útiles y eficientes en base a los recursos de hardware disponibles
- Estas abstracciones permiten el desarrollo de bibliotecas y aplicaciones que se ejecutarán sobre dicho sistema

## Características

## Características

- Mayormente codificado en C y assembly

## Características

- Mayormente codificado en C y assembly
- No tenemos *runtime library*. (~~printf~~, ~~fopen~~ ...)

## Características

- Mayormente codificado en C y assembly
- No tenemos *runtime library*. (~~printf~~, ~~fopen~~ ...)
- *Debuggear* es difícil

## Características

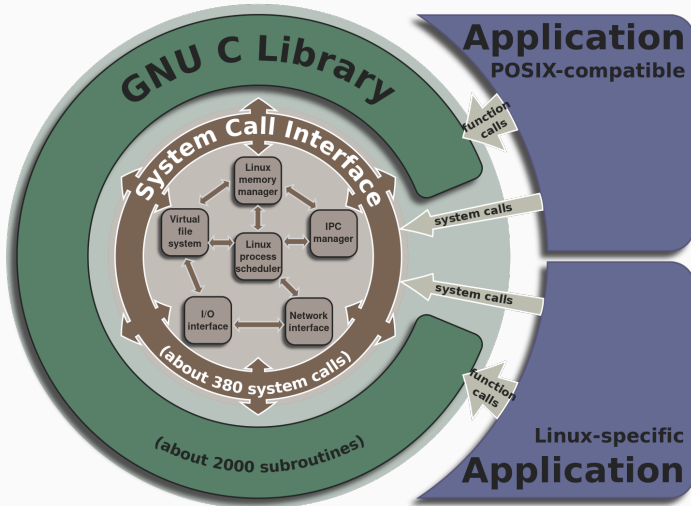
- Mayormente codificado en C y assembly
- No tenemos *runtime library*. (~~printf~~, ~~fopen~~ ...)
- *Debuggear* es difícil
- No hay un formato binario definido (no hay concepto de *archivo*)

## Características

- Mayormente codificado en C y assembly
- No tenemos *runtime library*. (~~printf~~, ~~fopen~~ ...)
- *Debuggear* es difícil
- No hay un formato binario definido (no hay concepto de *archivo*)

**Básicamente,  
¡todo lo que nos provee un Sistema Operativo no está!**





¿Qué vamos a estar viendo?

¿Qué vamos a estar viendo?

- Manejo de memoria: Segmentación y Paginación

¿Qué vamos a estar viendo?

- Manejo de memoria: Segmentación y Paginación
- Manejo de interrupciones y excepciones

¿Qué vamos a estar viendo?

- Manejo de memoria: Segmentación y Paginación
- Manejo de interrupciones y excepciones
- Conmutación de tareas

¿Qué vamos a estar viendo?

- Manejo de memoria: Segmentación y Paginación
- Manejo de interrupciones y excepciones
- Conmutación de tareas
- Políticas de acceso y protección

# Modo Real y Modo Protegido

---

Modos de operación:



Modos de operación:

- **Modo real:** modo en el que arrancan todos los x86, después de un power-up o reset.<sup>1</sup>

<sup>1</sup>Más información en *Intel 64 and IA-32 Architectures Software Developer's Manual*, Volumen 3, Capítulo 20: 8086 Emulation

Modos de operación:

- **Modo real:** modo en el que arrancan todos los x86, después de un power-up o reset.<sup>1</sup>
- **Modo protegido:** este modo es el estado nativo del procesador<sup>2</sup>

---

<sup>1</sup>Más información en *Intel 64 and IA-32 Architectures Software Developer's Manual*, Volumen 3, Capítulo 20: 8086 Emulation

<sup>2</sup>«*This mode is the native state of the processor*», *Ibíd.*, Volumen 3, Sección 2.2

Modo Real:

---

Modo Real:

- Trabaja por defecto en 16 bits

## Modo Real:

- Trabaja por defecto en 16 bits
- Podemos direccionar hasta 1MB de memoria<sup>1</sup>

---

<sup>1</sup>En realidad, se puede direccionar algo más de 1 MB, por medio de una técnica conocida

## Modo Real:

- Trabaja por defecto en 16 bits
- Podemos direccionar hasta 1MB de memoria<sup>1</sup>
- Los modos de direccionamiento son más limitados que en modo protegido

---

<sup>1</sup>En realidad, se puede direccionar algo más de 1 MB, por medio de una técnica conocida

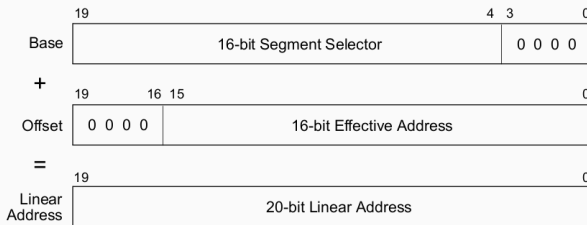
## Modo Real:

- Trabaja por defecto en 16 bits
- Podemos direccionar hasta 1MB de memoria<sup>1</sup>
- Los modos de direccionamiento son más limitados que en modo protegido
- No hay protección de memoria ni niveles de privilegio

---

<sup>1</sup>En realidad, se puede direccionar algo más de 1 MB, por medio de una técnica conocida

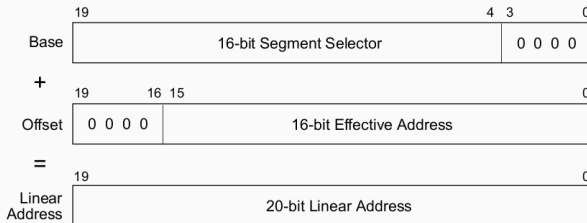
Las direcciones en modo real (20 bits) se forman con 2 componentes de 16 bits:





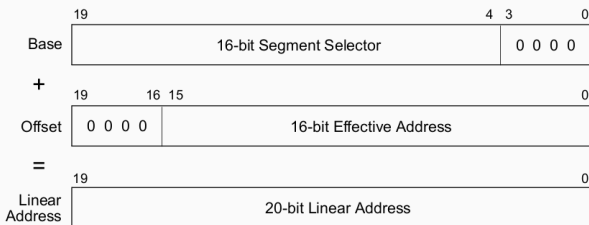
Las direcciones en modo real (20 bits) se forman con 2 componentes de 16 bits:

- **Dirección Base:** valor de un **registro de segmento** (CS,DS,ES,SS) shifteado 4 bits a la izquierda



Las direcciones en modo real (20 bits) se forman con 2 componentes de 16 bits:

- **Dirección Base:** valor de un **registro de segmento** (CS,DS,ES,SS) shifteado 4 bits a la izquierda
- **Offset:** el valor de un **registro** (AX, BX, CX, DX, SP, BP, SI y DI)



Ejemplo:

**Segmento : Offset**

0x12F3 : 0x4B27

Dirección Física =  $Segmento \times 16 + Offset$

0x17A57 =  $0x12F30 + 0x4B27$

	Modo real	Modo Protegido
<b>Memoria Disponible</b>	$\sim 1 \text{ MB}$	4 GB

	Modo real	Modo Protegido
Memoria Disponible	~ 1 MB	4 GB
Privilegios	:(	4 niveles de protección

	Modo real	Modo Protegido
<b>Memoria Disponible</b>	~ 1 MB	4 GB
<b>Privilegios</b>	: (	4 niveles de protección
<b>Interrupciones</b>	Rutinas de atención	Rutinas de atención con privilegios

	Modo real	Modo Protegido
<b>Memoria Disponible</b>	~ 1 MB	4 GB
<b>Privilegios</b>	:(	4 niveles de protección
<b>Interrupciones</b>	Rutinas de atención	Rutinas de atención con privilegios
<b>Set de instrucciones</b>	Todas	Depende el nivel de privilegio

¿Cómo vamos a trabajar?



¿Cómo vamos a trabajar?

- Vamos a usar **qemu** (emulador de x86)

¿Cómo vamos a trabajar?

- Vamos a usar **qemu** (emulador de x86)

¿Qué nos permite **qemu**?

¿Cómo vamos a trabajar?

- Vamos a usar **qemu** (emulador de x86)

¿Qué nos permite **qemu**?

- Utilizar instrucciones privilegiadas

¿Cómo vamos a trabajar?

- Vamos a usar **qemu** (emulador de x86)

¿Qué nos permite **qemu**?

- Utilizar instrucciones privilegiadas
- Ver estructuras del kernel (tablas de memoria, tablas de interrupciones, etc.)

¿Cómo vamos a trabajar?

- Vamos a usar **qemu** (emulador de x86)

¿Qué nos permite **qemu**?

- Utilizar instrucciones privilegiadas
- Ver estructuras del kernel (tablas de memoria, tablas de interrupciones, etc.)
- Bootear el procesador y pasarlo a modo protegido

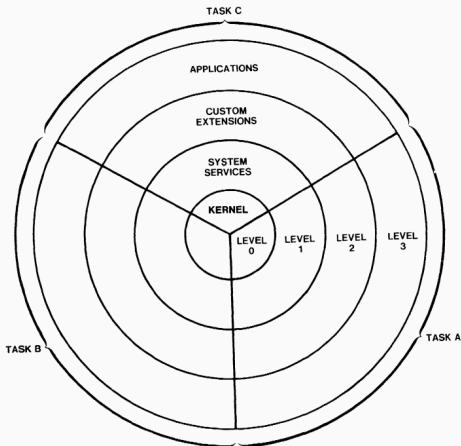
¿Cómo vamos a trabajar?

- Vamos a usar **qemu** (emulador de x86)

¿Qué nos permite **qemu**?

- Utilizar instrucciones privilegiadas
- Ver estructuras del kernel (tablas de memoria, tablas de interrupciones, etc.)
- Bootear el procesador y pasarlo a modo protegido
- Tener una ejecución controlada con posibilidad de *debugging*

¿Que tenía Intel<sup>®</sup> pensado?



## **Intervalo y consultas (5 minutos)**

---