

# Fundamentos de los Sistemas Operativos

## Tema 1. Conceptos generales ¿Qué es un sistema operativo?

© 2015 ULPGC - José Miguel Santos Espino

# Bibliografía para el Tema 1

- Texto principal
  - Fundamentos de los sistemas operativos (**Silberschatz**, 2006, 7ª ed.)  
**Capítulos 1 y 2**
- Textos alternativos
  - Sistemas Operativos: aspectos internos y principios de diseño (**Stallings**, 2005, 5ª ed.)  
**Capítulo 2**
  - Fundamentos de Sistemas Operativos. Teoría y ejercicios resueltos  
(Candela, García, Quesada, Santana, Santos, 2007)  
**Capítulo 1**  
*(o las ediciones originales en inglés)*



# ¿QUÉ ES UN SO?

# Primero: ¿qué hay en un sistema informático?

- **Hardware**
  - Procesador y memoria
  - Dispositivos de E/S: almacenamiento, red, HCI, impresión, etc.
- **Software**
  - Software de sistema: compilador, GUI, shell, etc.
  - Aplicaciones
- **Personas**
  - Usuarios en general
  - Administradores del sistema
  - Desarrolladores / programadores

# Y ahora: ¿qué es un SO?

- Un programa que actúa de intermediario entre los usuarios y el hardware
- Pertenece al *software del sistema*
- Objetivos:
  - Proveer un entorno para ejecutar las aplicaciones
  - Administrar eficientemente los recursos
  - Facilitar la interacción con el computador
  - Facilitar la evolución del software y del hardware

# Definiciones breves

1. Un **sistema** de software cuyo fin es que un sistema informático sea **operativo**.
2. Conjunto de programas que gestionan los recursos del sistema, optimizan su uso y resuelven conflictos.
3. Cualquier cosa que un fabricante de software te venda como un «sistema operativo».

# Los dos roles del SO

- **Interfaz con el hardware**
  - Añade características no existentes en el hw
  - Oculta características inconvenientes del hw
  - Ofrece una «máquina extendida»
- **Administrador de recursos**
  - Como si fuera un «gobierno del hardware»
  - Concede recursos de forma segura, justa y eficiente
  - No realiza trabajo productivo

# El SO como interfaz

- Visión: una «capa» que envuelve el hardware
- Ofrece una «máquina abstracta» con otras características
  - Oculta detalles incómodos del hardware
  - Amplía características no presentes en el hardware





# Interfaz del SO

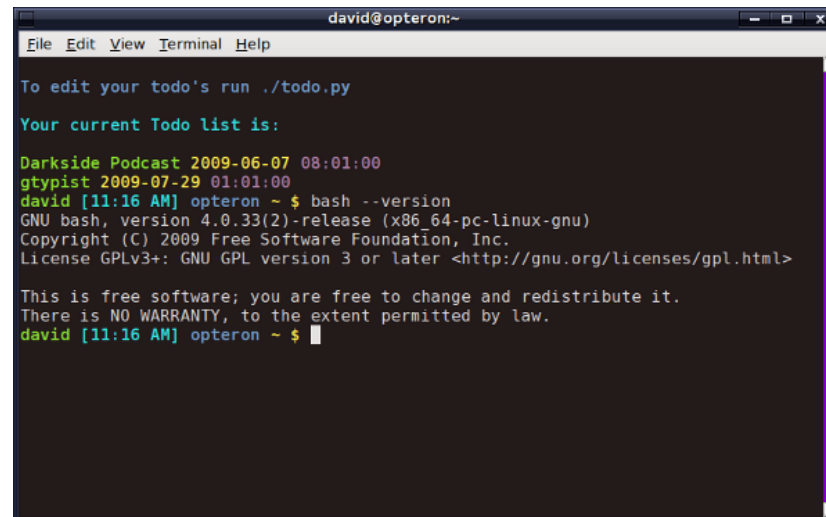
- ¿Para quién es la interfaz?
  - Usuarios en general → entorno de ejecución
  - Administradores → entorno de administración
  - Desarrolladores → interfaz de programación
- ¿Qué aspecto tiene la interfaz?
  - Texto (CLI = Command Line Interface)
  - Gráfica (GUI = Graphical User Interface)
  - Servicios de programación (API)

# Entorno de trabajo del SO

- El SO suele proporcionar utilidades básicas para que el usuario pueda realizar tareas comunes:
  - Trabajar con archivos y discos
  - Ejecutar aplicaciones → **cargador de programas**
  - Imprimir
  - Administrar el sistema: *backups*, usuarios...

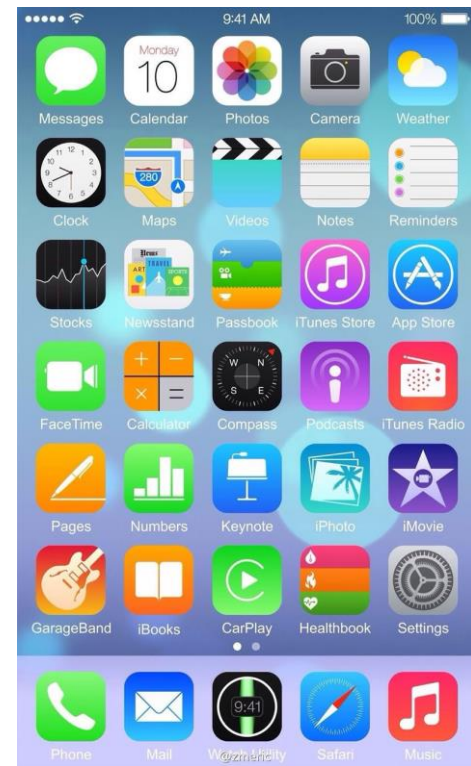
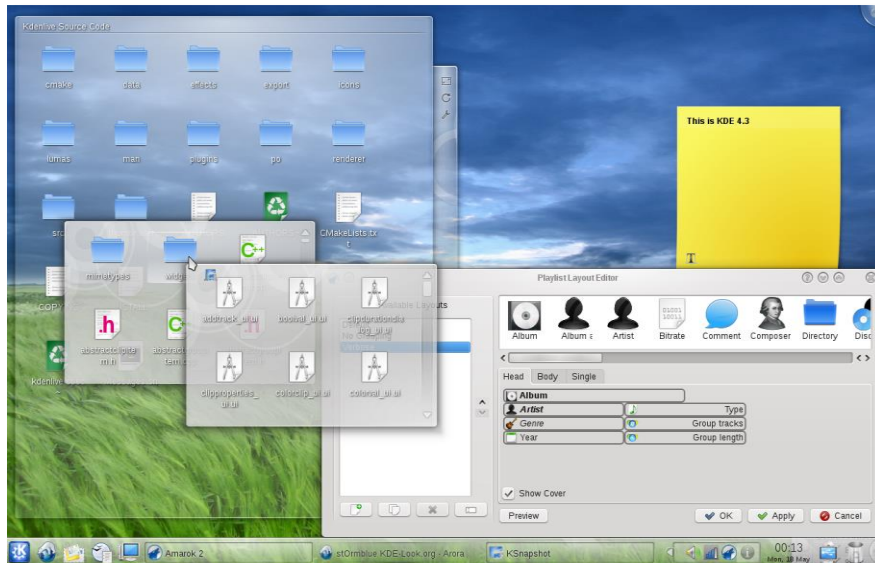
# CLI = Command Line Interface

- Incorpora un lenguaje sencillo para dar instrucciones al SO:
  - Cargar programas, trabajar con archivos, etc.
- Se le llama shell, intérprete de órdenes, consola...



```
david@opteron:~  
File Edit View Terminal Help  
To edit your todo's run ./todo.py  
Your current Todo list is:  
Darkside Podcast 2009-06-07 08:01:00  
gtypist 2009-07-29 01:01:00  
david [11:16 AM] opteron ~ $ bash --version  
GNU bash, version 4.0.33(2)-release (x86_64-pc-linux-gnu)  
Copyright (C) 2009 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software; you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
david [11:16 AM] opteron ~ $
```

# GUI = Graphical User Interface




# API del SO: llamadas al sistema

- API = Application Programming Interface
- El SO ofrece a los desarrolladores y a los procesos un conjunto de servicios públicos, accesibles mediante una API  
→ **llamadas al sistema (*system calls*)**


# Llamadas al sistema: ejemplo

Llamada **write()** de UNIX. Escribe un bloque de datos en un fichero o en un dispositivo de E/S.


```
int write ( int fd, void* buffer, size_t size )
```




Resultado:  
número de bytes  
escritos



Descriptor de fichero  
(identifica el fichero  
donde vamos a escribir)



Apuntador a la zona  
de memoria donde  
están los datos



Longitud de los  
datos en bytes

# Beneficios

*¿Qué ganamos interponiendo esta interfaz entre los programas y el hardware?*

- **Usabilidad** (la interfaz es más cómoda que el hw)
- **Seguridad** (se ocultan vulnerabilidades del interior del hardware)
- **Portabilidad** (independencia del hardware)
- **Interoperabilidad** (podemos compartir información con otros sistemas que usen la misma interfaz)
- **Mantenibilidad** (podemos hacer mejoras o adaptaciones dentro del SO sin obligar a hacer cambios en los programas de usuario)
- **Productividad** (por todo lo anterior)

# El SO como administrador de recursos

- Procesos y recursos
  - **Proceso**: programa en ejecución
  - **Recurso**: algo físico o virtual que necesita un proceso para ejecutarse
- Los recursos son escasos → los procesos compiten por ellos
- El SO actúa como árbitro/mediador, que asigna recursos de forma **justa y eficiente**



# El SO como administrador de recursos

- El SO debe determinar a quién se le entregan los recursos, qué cantidad, en qué momento y por cuánto tiempo.

→ **políticas de gestión de recursos**



# El SO como administrador de recursos

- Criterios que deben cumplir las políticas del SO:
  - optimizar el **rendimiento** del sistema
  - **justicia** en el reparto → evitar acaparamientos e *inanición* de los procesos perjudicados
  - garantizar la **seguridad** del sistema (confidencialidad, integridad, disponibilidad)
- Estos criterios entran en conflicto
  - Ej. no se puede dar el máximo rendimiento y al mismo tiempo dar un reparto justo

# Seguridad: los tres elementos

- **CIA = Confidentiality + Integrity + Availability**
- **Confidencialidad** → intimidad, privacidad, etc.
- **Integridad** → que la información no se corrompa
- **Disponibilidad** → que el sistema continúe prestando servicio

# GRANDES LOGROS DEL S.O.

# Algunos logros históricos de los SO

- Interfaz uniforme con la E/S
- Multiprogramación
- Memoria paginada
- Memoria virtual
- Sistemas de archivos
- Control del acceso concurrente
- Protección y seguridad

# Interfaz uniforme con la E/S

- La E/S es tremendamente diversa
- Ej. almacenamiento: óptico, disco magnético, SSD, cinta magnética...
- Cada clase de dispositivo se programa de forma distinta:
  - Tamaño de la unidad de transferencia de datos
  - Protocolo de comunicación (síncrono, asíncrono...)
  - Codificación de la información
  - Control de errores
  - ...

# Interfaz uniforme con la E/S

- Solución del SO: ofrecer a los desarrolladores una API uniforme para acceder a cualquier dispositivo de E/S, ej.:

```
readIO ( int device_id, void* data, int length )  
writeIO ( int device_id, void* data, int length )
```

- Para cada clase de dispositivo existe una implementación de esta API
  - manejador de dispositivo (*device driver*)
  - Interno del SO, es transparente para el usuario final

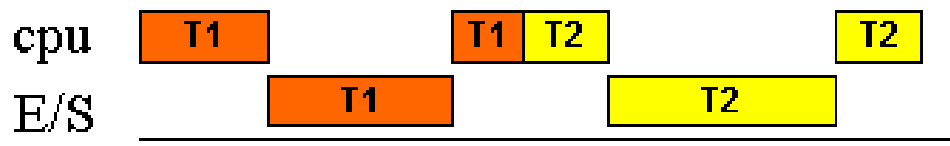
# Interfaz uniforme con la E/S

- ¿Qué conseguimos? **independencia del dispositivo**
  - Abstraemos los detalles de implementación de cada clase de periférico
  - Ganamos en portabilidad (el mismo código sirve para dispositivos diferentes)
  - Adaptación a futuras clases de periféricos
  - Podemos prohibir el acceso directo a la E/S (sólo trabajar con la API del SO) → más seguridad

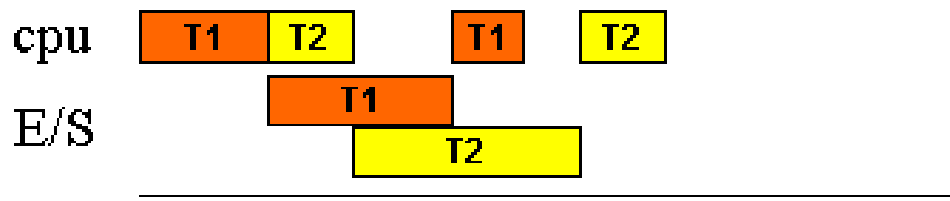


# Multiprogramación

- También llamada **multitarea** (*multitasking*)
- Cuando un proceso se bloquea al esperar por la E/S, ejecutamos en la CPU instrucciones de otro proceso.
- Los procesos entrelazan su ejecución: **conurrencia**.
- La CPU y la E/S trabajan a la misma vez  $\Rightarrow$  se terminan más trabajos en menos tiempo



*Sin multiprogramación*



*Con multiprogramación*

# La memoria en un sistema multiprogramado



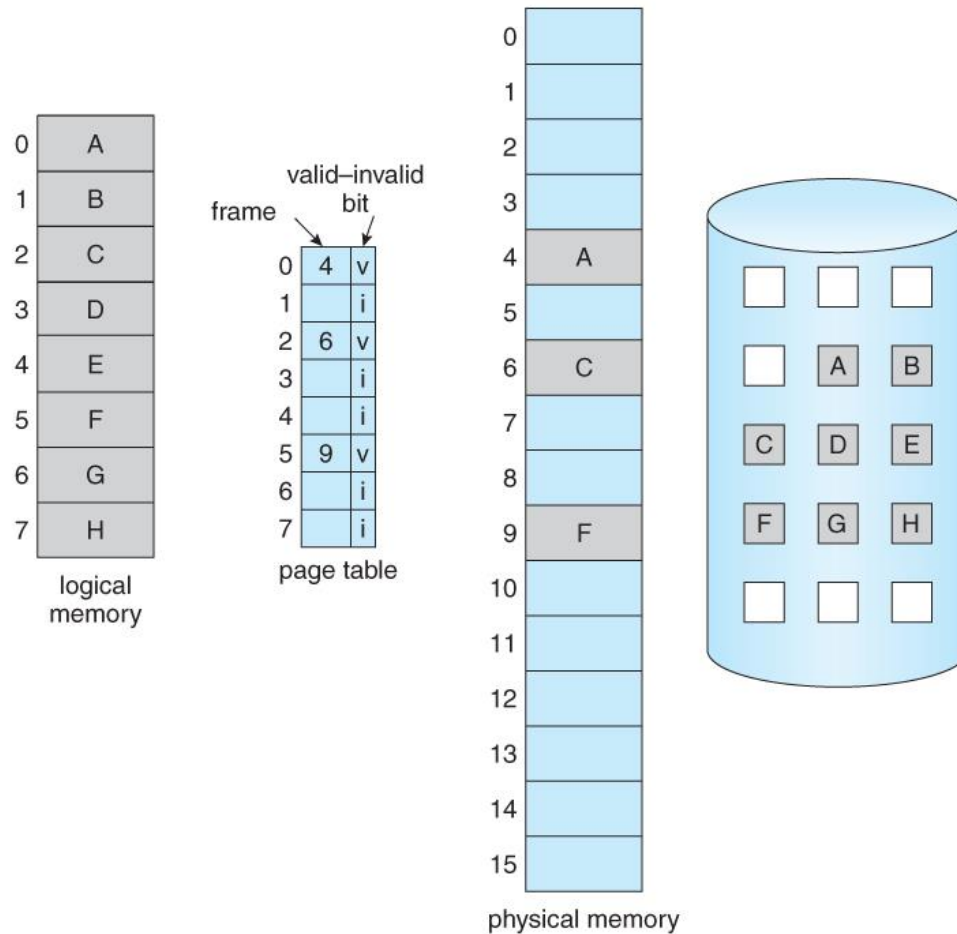
# Multiprogramación

- Cuestiones que surgen en un sistema multiprogramado (y que no existen en un sistema sin multiprogramación):
  - Cuando el procesador queda libre, ¿a qué proceso elegimos? → **planificación de CPU**
  - Competencia por el consumo de la memoria
  - Protección de las zonas privadas de memoria
  - Conflictos ante el acceso simultáneo a los recursos
  - Riesgo de **interbloqueo** (*deadlock*)

# Paginación y memoria virtual

- **Paginación.** Podemos trocear un programa en pequeñas «páginas» que se pueden colocar en zonas diferentes de la memoria.
- **Memoria virtual.** El programa no necesita estar cargado totalmente en memoria. El almacén secundario se usa como extensión de la memoria principal.
- Todo se resuelve automáticamente, sin que el usuario ni el programador tengan que intervenir.

# Paginación y memoria virtual



(Silberchatz, Galvin, Gagne, 2013)

# Sistemas de archivos

- Usuarios → trabajan con documentos, programas, imágenes, música...
- Almacenamiento físico → trabaja con bloques de datos de tamaño fijo.
  - Extraño para los usuarios
- Creamos un recurso virtual, llamado «archivo», que tiene nombre y contiene información.
- También el concepto «carpeta» o «directorio» para organizar los archivos.
- Abstracción muy útil del almacenamiento físico.

# Control del acceso concurrente

- ¿qué pasa si varios procesos intentan trabajar al mismo tiempo con un recurso que no se puede compartir?
  - Ej. una impresora
- El SO debe conocer quién está usando el recurso y forzar la espera si está ocupado
- No es trivial resolver este problema de forma segura y eficiente...  
*(la solución, en el Tema 3)*

# Protección y seguridad

- ¿Cómo evitamos que un proceso dañe la memoria ocupada por otro proceso, o por el SO?
- ¿Cómo impedimos que un proceso acceda directamente a la E/S, saltándose las políticas de asignación de recursos?
- ¿Cómo garantizamos que los datos privados de un usuario no pueden ser leídos por cualquier proceso?
- ¿Cómo nos aseguramos de que sólo las personas autorizadas pueden acceder al sistema?



# **ALGUNOS TIPOS DE SISTEMAS**

# Entornos de computación

- Ordenadores personales
- Dispositivos de mano: móviles y tabletas
- Sistemas empotrados (*embedded systems*)
- Servidores + multiprocesadores
- *Clusters* de servidores
- Supercomputadores
- Sistemas distribuidos
- Sistemas virtualizados
- Sistemas en la nube (*clouds*)

# Algunos tipos de sistemas

- Procesamiento por lotes (*batch processing*)
- Tiempo compartido (*time sharing*)
- Tiempo real (*real time*)
- Sistemas multiusuario
- Máquinas virtuales

# Sistemas de procesamiento por lotes (*batch processing*)

- Históricamente, fueron los primeros SO (principios de los 1950)
- Objetivo: automatizar la ejecución de trabajos y aumentar la utilización del procesador
- Los trabajos se agrupaban en **lotes** que se iban ejecutando en secuencia
- Los más primitivos eran secuenciales; la multiprogramación se incorporó a finales de los 50
- Primer lenguaje para dictar tareas al SO  
→ JCL (Job Control Language)

# Sistemas de tiempo compartido (*time sharing*)

- Inventados en los 1950, comercializados en los 60.
- Avance sobre los sistemas por lotes, para conseguir interactividad.
- Cada proceso dispone de una pequeña rodaja de tiempo periódica. Los procesos se van turnando en la CPU.
- Si la rodaja de tiempo es lo bastante pequeña (milisegundos), el usuario no percibe las pausas periódicas de su sesión.

# Sistemas de tiempo real

## *(real time systems)*

- Diseñados para cumplir tareas que deben completarse en un plazo prefijado (sistemas de control industrial, sistemas multimedia...).
- Usan algoritmos de planificación de procesador especiales.
- **S.T.R. crítico** → para industria y sistemas empujados en los que el cumplimiento de plazos es crítico. Suelen prescindir de servicios que afectan a los tiempos (ej. Memoria virtual).

# Sistemas multiusuario

- Un sistema **multiusuario** reconoce que hay varios perfiles de acceso, con privilegios distintos:
  - Permisos de acceso a ficheros y aplicaciones
  - Cuotas de espacio o de tiempo de procesador
  - Prioridad en el acceso a los recursos
  - ...
- Ojo: multiusuario  $\neq$  multitarea (puede haber sistemas multitarea que no son multiusuario)

# Multiprocesadores

- Desde unos pocos hasta miles de procesadores
- Varios modelos de acceso a memoria
  - UMA – memoria compartida
  - NUMA – no compartida
- Varios modelos de ejecución de procesos:
  - **SMP** – multiprocesamiento simétrico → una tarea se puede ejecutar en cualquier procesador
  - **AMP** - Multiprocesamiento asimétrico → hay especialización de tareas (ej. un procesador ejecuta el SO y otro los procesos de usuario)



# Sistemas distribuidos

- Un S.D. es un conjunto de computadores conectados en red y que se utiliza como si fuera un único sistema con múltiples procesadores + una gran memoria compartida + un gran almacenamiento secundario.
- No existe un «sistema distribuido universal», pero sí hay servicios con características de sistema distribuido.
  - Servicios en la nube (Dropbox, Amazon...)
  - La WWW

# Máquinas virtuales

- Emulación por software de una máquina física.
- Sobre la máquina virtual pueden ejecutarse programas implementados para la máquina física emulada.
- Ventaja: no necesitamos el sistema original
- Inconveniente: la emulación es más lenta

# Ejemplos de máquinas virtuales

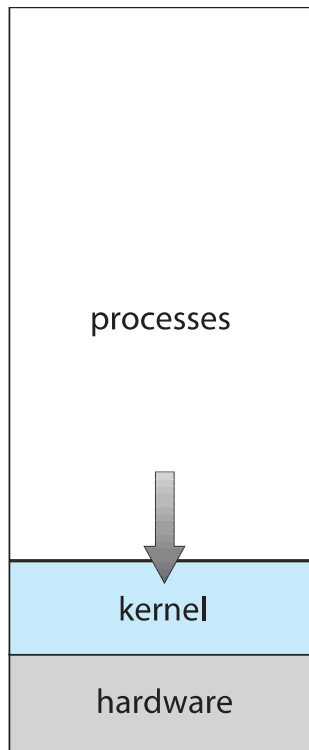
- **IBM VM:** (años 1960) sobre un sistema por lotes, ofrecía a cada usuario su propia máquina virtual no multiprogramada; las m.v. se planificaban con tiempo compartido.
- **Java:** los programas compilados en Java corren sobre una máquina virtual (JVM).
- **VMware:** capaz de ejecutar al mismo tiempo varias sesiones Windows, Linux, Mac OS X, etc. sobre plataforma PC o Mac.

# Usos de las máquinas virtuales

- Crear entornos protegidos: cada máquina virtual está aislada de las otras
- Independencia de la plataforma (ej. Java)
- Pervivencia de sistemas antiguos (ej. emuladores MSDOS, consolas de juegos...)
- Desarrollo: se pueden escribir y probar aplicaciones para un hardware que no tenemos

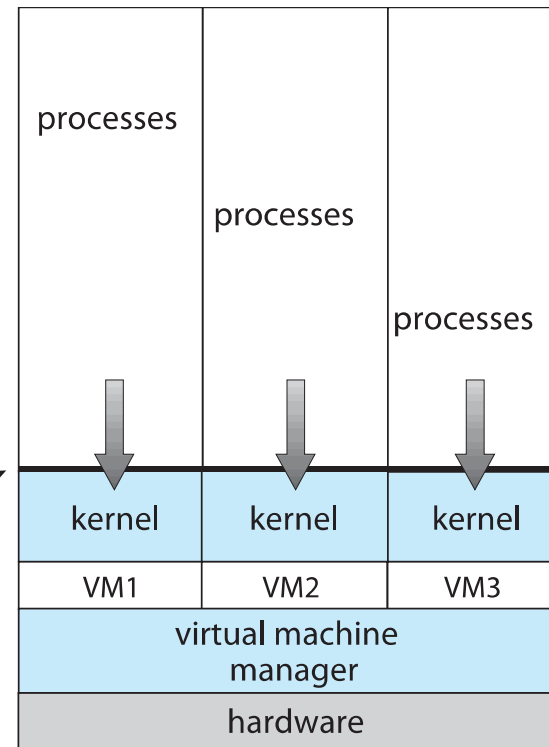
# VMM (virtualización): VMWare, VirtualBox, Parallels...

**Sin virtualización**



(a)

**Con virtualización**



(b)

# Tema 1

## FIN de la primera parte

© 2015 ULPGC – José Miguel Santos Espino