

# Sistemas Operativos

o ¿cómo funciona un computador?, parte 2

IIC1005 - Computación: Ciencia y Tecnología del Mundo Digital

Cristian Ruz – [cruz@ing.puc.cl](mailto:cruz@ing.puc.cl)

Departamento de Ciencia de la Computación  
Pontificia Universidad Católica de Chile

Semestre 1-2018

# Contenidos

# ¿Qué tenemos?

¿Qué tenemos?

- Hardware . . .

¿Y hacia dónde vamos?

# ¿Qué tenemos?

¿Qué tenemos?

- Hardware . . . Y un set de instrucciones

¿Y hacia dónde vamos?

# ¿Qué tenemos?

¿Qué tenemos?

- Hardware . . . Y un set de instrucciones

¿Y hacia dónde vamos?

- A ejecutar un programa

# Hay que hablar en el lenguaje correcto

- Nosotros decimos:

```
int main() {  
    int a,b,c;  
    a = 2;  
    b = 3;  
    c = a+b;  
    printf("%d\n",c);  
    return 0;  
}
```

- Pero el computador sólo entiende:

```
movl $2, -8(%rbp)  
movl $3, -12(%rbp)  
movl -8(%rbp), %eax  
addl -12(%rbp), %eax  
movl %eax, -16(%rbp)  
movl -16(%rbp), %esi  
movb $0, %al  
callq _printf
```

Necesidad: efectuar esta traducción

# ¿Cómo usar el computador?

Necesitamos ser capaces de usar:

CPU

Para ejecutar programas

Memoria

Para almacenar y leer programas

I/O

Para interactuar con el ambiente

# ¿Cómo usar el computador?

Pero tenemos:



Para usarla debemos escribir cada instrucción



# ¿En serio?

Antes de ejecutar:

```
movl    $2, -8(%rbp)
movl    $3, -12(%rbp)
movl    -8(%rbp), %eax
addl    -12(%rbp), %eax
movl    %eax, -16(%rbp)
movl    -16(%rbp), %esi
movb    $0, %al
callq   _printf
```

Necesitamos decir al computador:

- Qué hacer cuando se apreta una tecla, o se mueve el mouse.
- Cómo desplegar un carácter o imagen en pantalla
- En qué posición de la memoria podemos escribir
- Cómo leer archivos desde el disco

MUCHOS programas requieren estos mismos servicios

# ¿Y ahora quién podrá ayudarnos . . . ?

Un programa “maestro” que se ejecuta permanentemente y permite:

- Entender entradas de teclado, mouse
- Desplegar caracteres, imágenes en pantalla
- Leer y escribir desde ciertos lugares de memoria
- ... y en suma ... **ejecutar programas**

Necesitamos un

## SISTEMA OPERATIVO

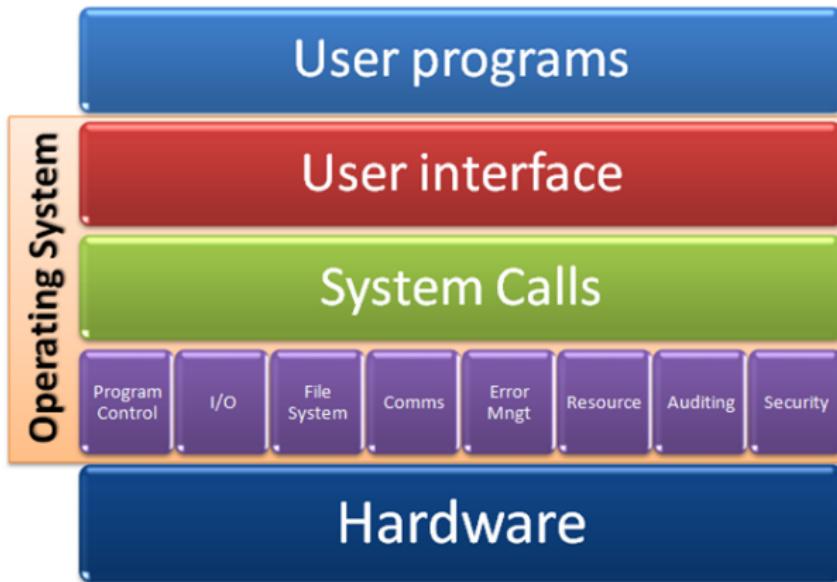
# Entonces ... ¿qué es un sistema operativo?

Programa que...

- Permite ejecutar programas (¿“procesos”?) sobre la CPU
- Administrar la memoria disponible entre uno o más procesos
- Administrar dispositivos de I/O para que puedan ser usados por los procesos

**Permite utilizar el computador**

# ¿Qué es un sistema operativo?

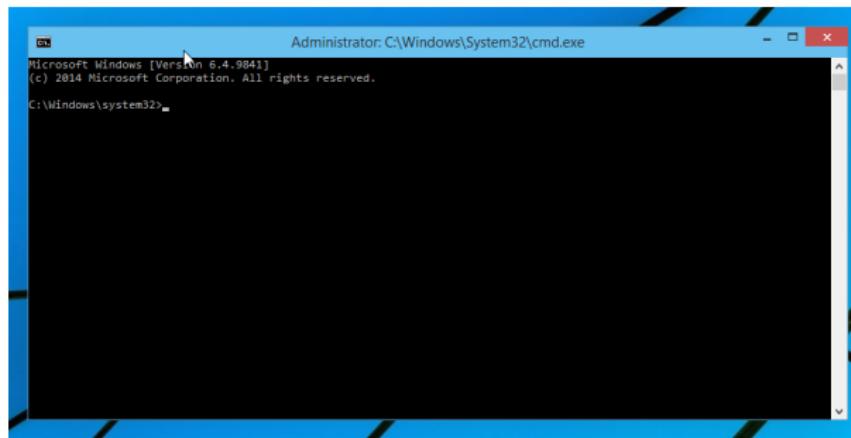


# Interfaces de Usuario (UI)

## Command-Line (línea de comandos)

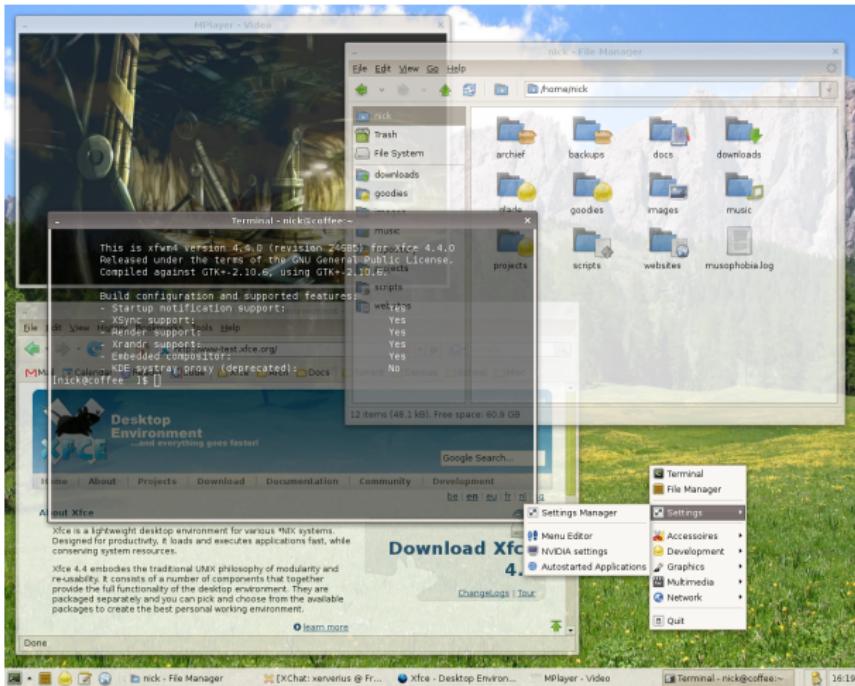
También conocidos como **shells**

```
cruz@hercules:~$ cd iic2333/  
cruz@hercules:iic2333$ ls  
examples  
cruz@hercules:iic2333$ cd examples/  
cruz@hercules:examples$ ls  
fork2.cpp fork3.c fork.c shmem-cons.c shmem-prod.c waitZombies.c  
cruz@hercules:examples$ █
```



# Interfaces de Usuario (UI)

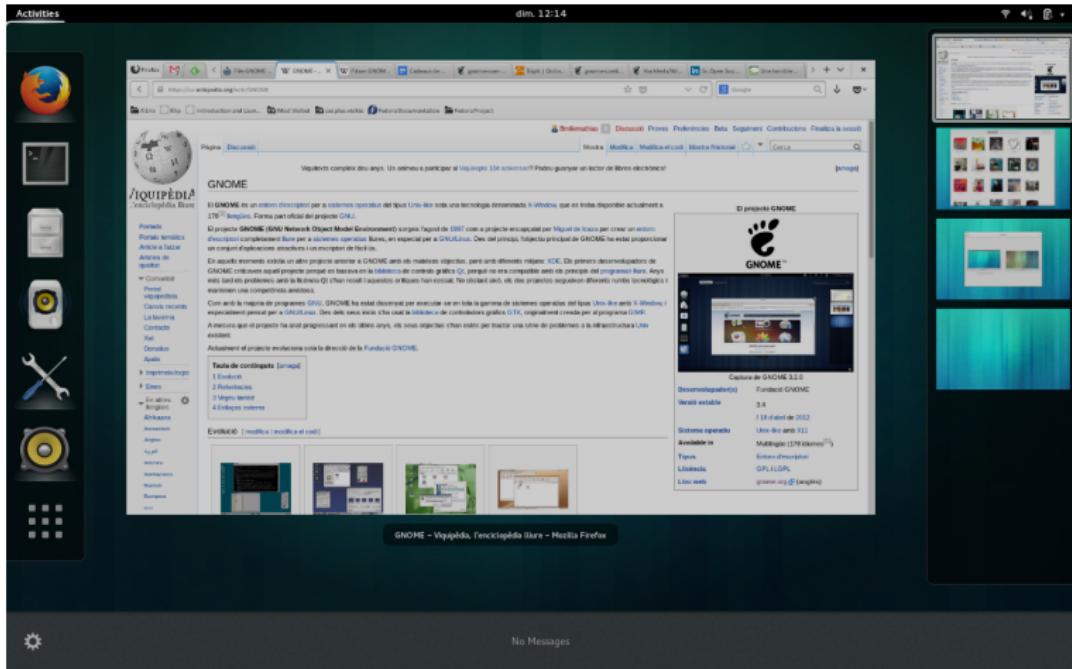
## Graphical User Interface (GUI)



Xfce

# Interfaces de Usuario (UI)

## Graphical User Interface (GUI)



## GNOME

# Interfaces de Usuario (UI)

## Graphical User Interface (GUI)



GNOME

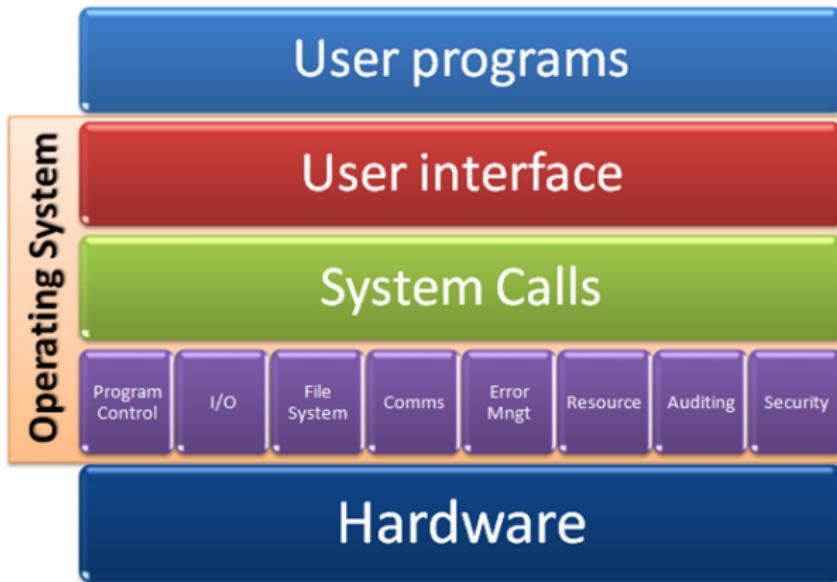
# Interfaces de Usuario (UI)

## Graphical User Interface (GUI)



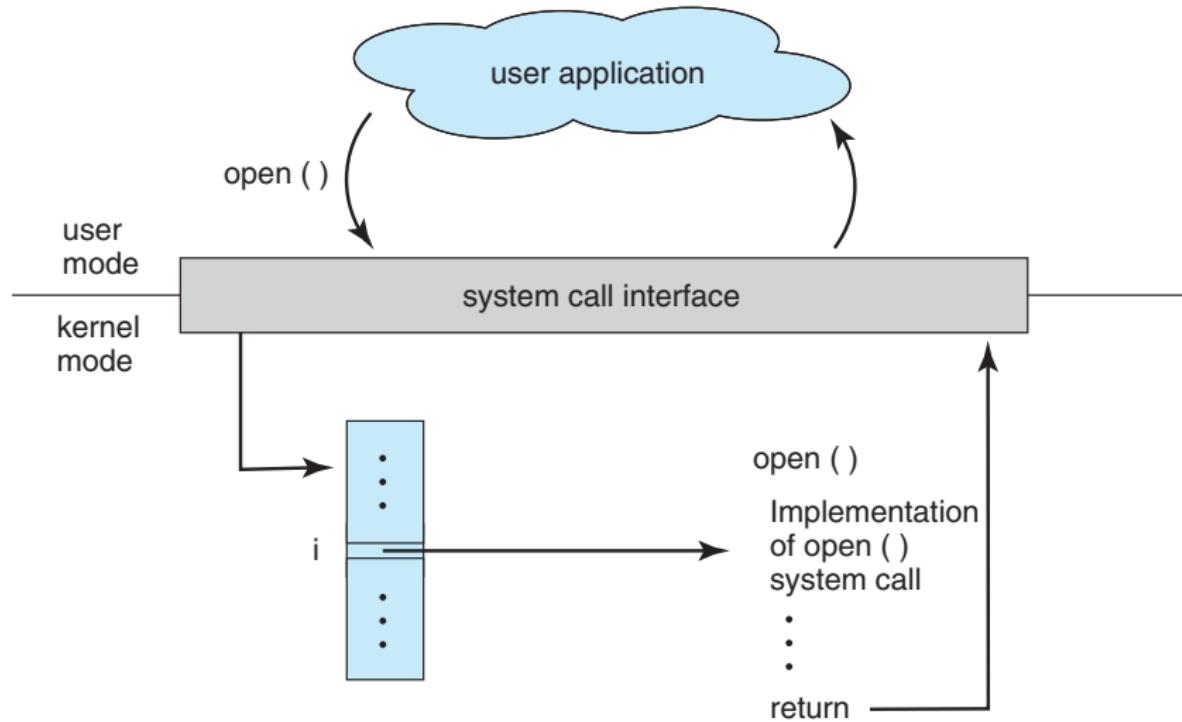
GNOME

# ¿Qué es un sistema operativo?



# Llamadas al sistema

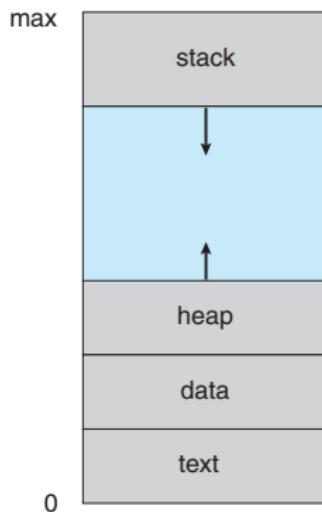
Programas utilizan llamadas al sistemas frecuentemente



# ¿Qué hay en un proceso?

What's in a process

- **Código** ≡ texto . . . información estática
- **Program Counter + registros**, . . . actividad actual
- **Stack**, datos temporales
  - Parámetros de función, direcciones de retorno, variables locales
- **Datos**, variables globales
- **Heap**, memoria asignada dinámicamente (en tiempo de ejecución)



# Muchos procesos quieren ocupar la CPU

Pero solo uno (o unos) puede(n)

The image shows two screenshots illustrating system resource usage.

**Top Command Output:**

```
top - 15:21:18 up 3 min, 2 users, load average: 1.24, 1.44, 0.65
Tasks: 146 total, 1 running, 145 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.8%us, 0.7%sy, 0.0%ni, 97.5%id, 0.0%hi, 0.0%si, 0.0%st
Mem: 4058812k total, 1244340k used, 2814472k free, 154292k buffers
Swap: 3903756k total, 0k used, 3903756k free, 637368k cached

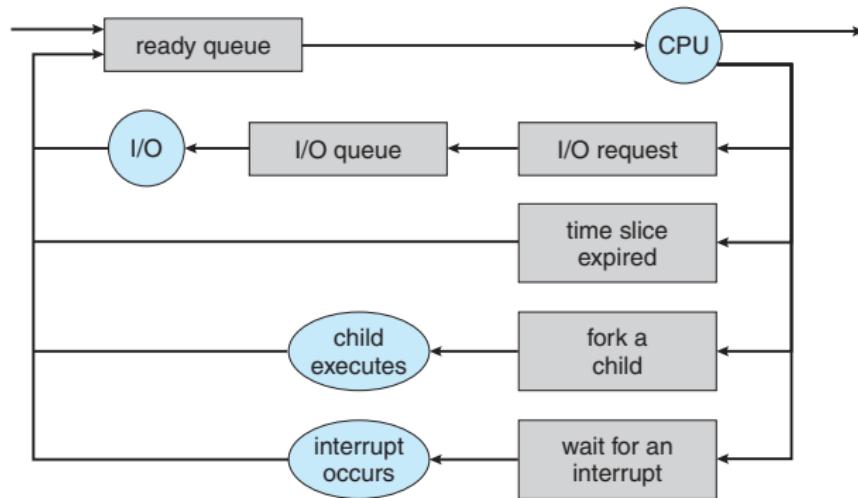
PID USER PR NI VIRT RES SHR %CPU %MEM TIME+ COMMAND
1164 root 20 0 164m 51m 22m S 2 1.3 0:04.92 Xorg
2533 rob 20 0 450m 13m 9m S 1 0.3 0:00.17 xfce4-screensho...
1921 rob 20 0 175m 11m 8904 S 0 0.3 0:00.19 xfwm4
1925 rob 20 0 564m 17m 12m S 0 0.4 0:00.46 xfdesktop
1958 root 20 0 45184 800 448 S 0 0.0 0:00.01 udisks-daemon
1976 rob 20 0 434m 10m 8492 S 0 0.3 0:00.21 xfce4-cpufreq-p...
2350 rob 20 0 1004m 128m 72m S 0 3.1 0:03.07 soffice.bin
2427 rob 20 0 439m 13m 10m S 0 0.4 0:00.12 xfce4-terminal
2532 rob 20 0 19368 1316 948 R 0 0.0 0:00.04 top
1 root 20 0 24144 2224 1324 S 0 0.1 0:00.65 init
2 root 20 0 0 0 0 S 0 0.0 0:00.00 kthreadread
3 root 20 0 0 0 0 S 0 0.0 0:00.08 ksftirqd/0
4 root 20 0 0 0 0 S 0 0.0 0:00.20 kworker/0:0
5 root 20 0 0 0 0 S 0 0.0 0:00.29 kworker/u:0
6 root RT 0 0 0 0 S 0 0.0 0:00.00 migration/0
7 root RT 0 0 0 0 S 0 0.0 0:00.00 migration/1
8 root 20 0 0 0 0 S 0 0.0 0:00.24 kworker/1:0
```

**Windows Task Manager:**

Name	PID	Status	User name	CPU	Memory ..	Description
ApplicationFrameHost.exe	6872	Running	Russell	00	9,956 K	Application Frame Host
audiodg.exe	19756	Running	LOCAL SERVICE	00	4,856 K	Windows Audio Device Graph Isolation
cssrs.exe	620	Running	SYSTEM	00	708 K	Client Server Runtime Process
cssrs.exe	32204	Running	SYSTEM	00	1,252 K	Client Server Runtime Process
daemonui.exe	1044	Running	UpdatasUser	00	2,512 K	NVIDIA Settings Update Manager
dashHost.exe	2932	Running	LOCAL SERVICE	00	5,672 K	Device Association Framework Provider Host
dllhost.exe	5468	Running	SYSTEM	00	1,124 K	COM Surrogate
downloader2.exe	10488	Running	SYSTEM	00	1,124 K	File Downloader
diagram.exe	20548	Running	DWM-2	00	48,624 K	Desktop Window Manager
explorer.exe	11696	Running	Russell	00	78,556 K	Windows Explorer
FlashUtil.ActiveX.exe	29404	Running	Russell	00	2,272 K	Adobe® Flash® Player Utility
explorer.exe	34732	Running	Russell	00	34,280 K	Internet Explorer
explorer.exe	19916	Running	Russell	00	103,824 K	Internet Explorer
explorer.exe	8152	Running	Russell	00	52,740 K	Internet Explorer
explorer.exe	27740	Running	Russell	00	42,232 K	Internet Explorer

# Colas de *scheduling*

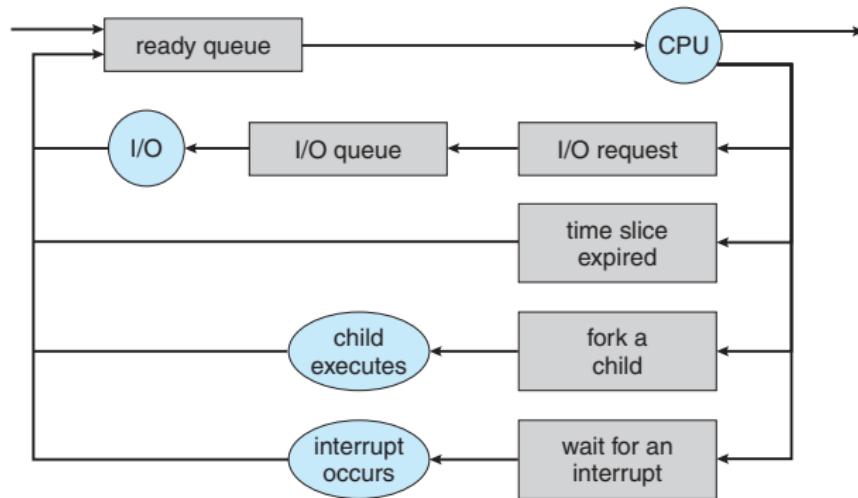
Procesos se *encolan* para ocupar la CPU



¿Bajo qué criterio?

# Colas de *scheduling*

Procesos se *encolan* para ocupar la CPU



¿Bajo qué criterio? → **algoritmos de scheduling**

# Memoria

## Memoria

- Un gran arreglo de *bytes*
  - Cada uno con su propia **dirección**
- 
- PC lee instrucciones desde una dirección de memoria
  - Instrucciones pueden requerir leer operandos desde otra dirección de memoria
  - Resultados suelen ser almacenados en alguna dirección de memoria

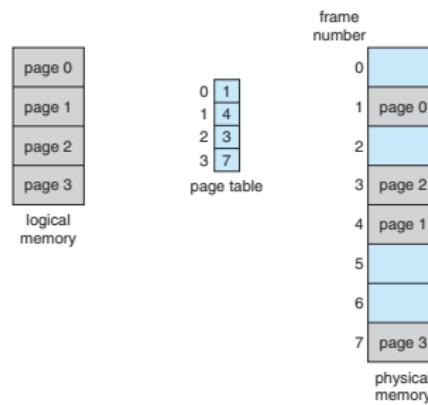
# Paginación

División de memoria en unidades de tamaño fijo:

- Memoria física se divide en **frames** (marcos)
- Memoria lógica se divide en **páginas**

Dirección de memoria paginada contiene:

$\langle \text{numeroDePagina}, \text{offset} \rangle$



# Ejemplo de paginación

Páginas de  $2^n$  bytes,  $n = 2$ . Direcciones lógicas de  $m$ -bit,  $m = 4$ .  
Memoria física de 32 bytes == 8 páginas.

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

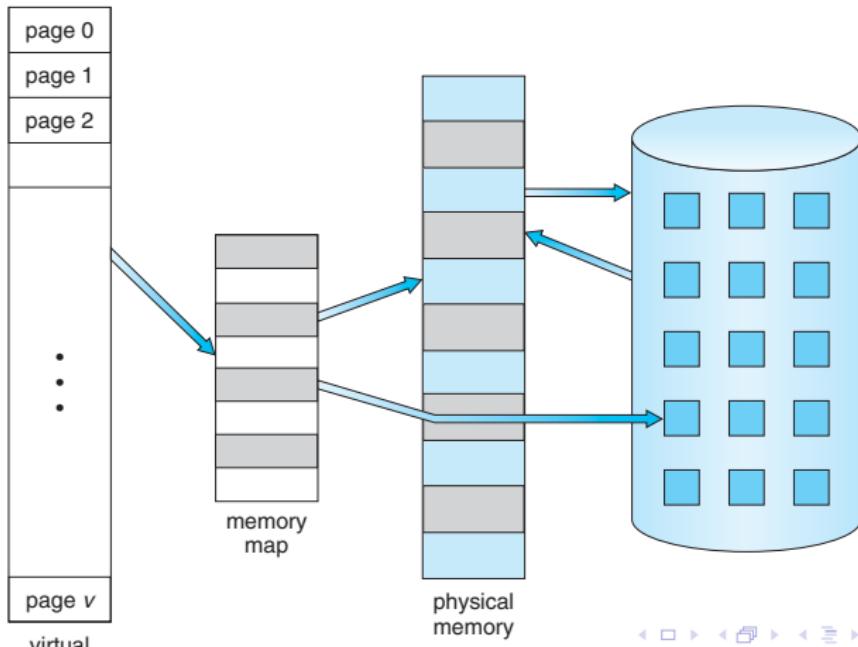
0	5
1	6
2	1
3	2

page table

0	
4	i
5	j
6	k
7	l
8	m
9	n
10	o
11	p
12	
16	
20	a
21	b
22	c
23	d
24	e
25	f
26	g
27	h
28	

# Memoria Virtual: Funcionamiento

**Memoria Virtual** aumenta la separación entre la memoria visible por el programador, y la memoria física de la máquina en que se ejecuta el programa.



## Archivo

Colección de información en almacenamiento secundario

- Unidad de almacenamiento para el usuario
- Agrupación lógica de *bytes*

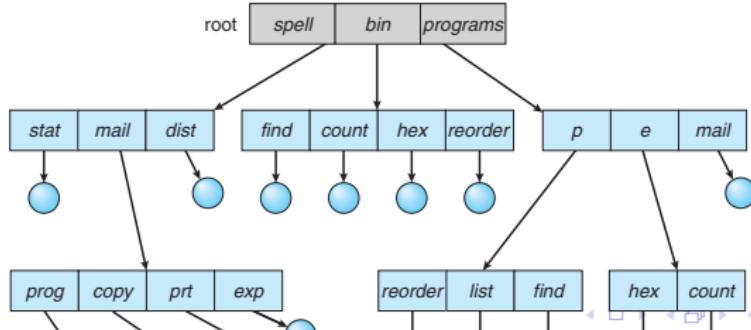
Contenido: cualquier cosa

- Significado está determinado por la forma en que se utiliza su contenido
- Texto, imagen, código fuente, código binario (ejecutable), ...

# Directarios

¿Cómo encontrar archivos?

- Sistemas de archivos mantienen *listas* de archivos y sus ubicaciones en bloques de disco
- Archivos se listan en una **tabla de contenidos** ó **árbol de directorios**
- Camino desde la raíz al directorio: **ruta** ó **path**
- Cada proceso tiene un directorio actual: **current directory** (pwd)
- **Ruta absoluta:** Ruta desde la raíz. Ej: /spell/mail/exp
- **Ruta relativa:** Ruta desde la ubicación actual (*current directory*). Ej: p/list solo existe si el directorio actual es /programs/



# Sistemas de archivos

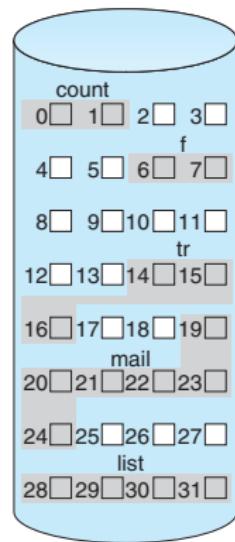
Un **sistema de archivos** permite obtener una dirección de disco, a partir de un datos simbólicos: nombre de archivo, directorio, rutas, links, ...  
Algunos ejemplos:

- ISO9660 (CD-ROMs)
- UFS: Unix File System, basado Berkeley Fast File System (FFS)
- FAT, FAT32, NTFS: Windows
- ext3, ext4: Linux
- GFS, FUSE

# Asignación de bloques

## Asignación contigua

- Archivo de  $n$  bloques, bloque inicial  $b$
- Bloques:  $b, b + 1, \dots, b + n - 1$
- Fácil de acceder a siguiente bloque

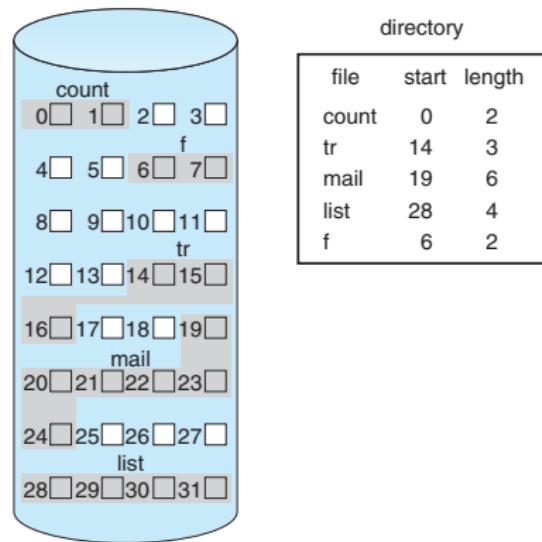


directory		
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

# Asignación de bloques

## Asignación contigua

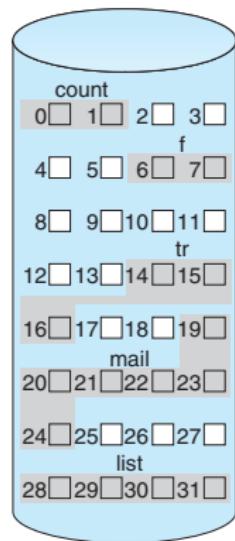
- Archivo de  $n$  bloques, bloque inicial  $b$
- Bloques:  $b, b + 1, \dots, b + n - 1$
- Fácil de acceder a siguiente bloque
- -Problema: fragmentación externa



# Asignación de bloques

## Asignación contigua

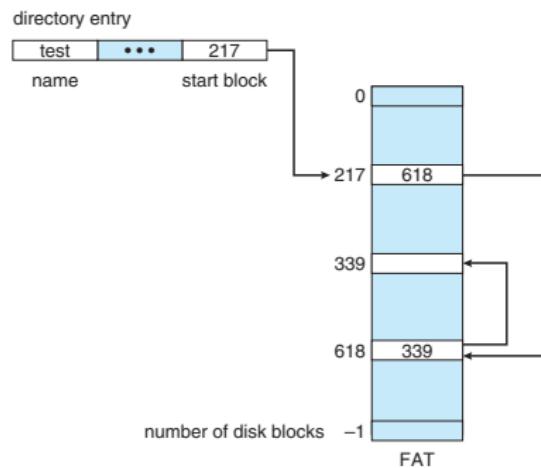
- Archivo de  $n$  bloques, bloque inicial  $b$
- Bloques:  $b, b + 1, \dots, b + n - 1$
- Fácil de acceder a siguiente bloque
- -Problema: fragmentación externa
- -Require operaciones de **compactación** (defragmentación)



# Asignación de bloques

Asignación enlazada (*linked*): FAT

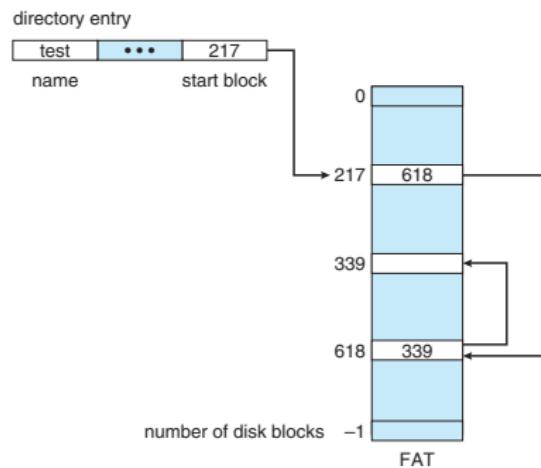
- **FAT: File Allocation Table**
- Tabla al inicio del disco con una entrada por archivo
- Último cluster almacena EOF (*end-of-file*)



# Asignación de bloques

Asignación enlazada (*linked*): FAT

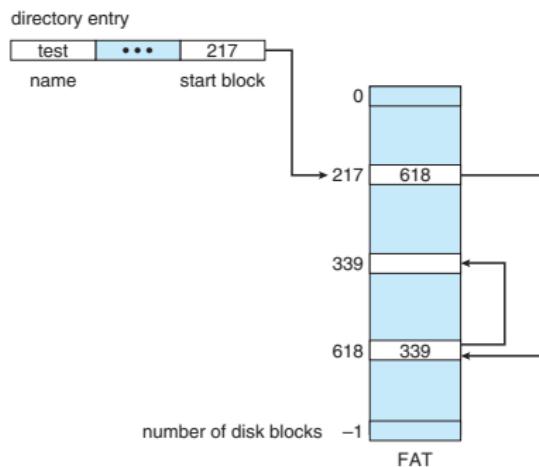
- **FAT: File Allocation Table**
- Tabla al inicio del disco con una entrada por archivo
- Último cluster almacena EOF (*end-of-file*)
- +Mejor acceso aleatorio



# Asignación de bloques

Asignación enlazada (*linked*): FAT

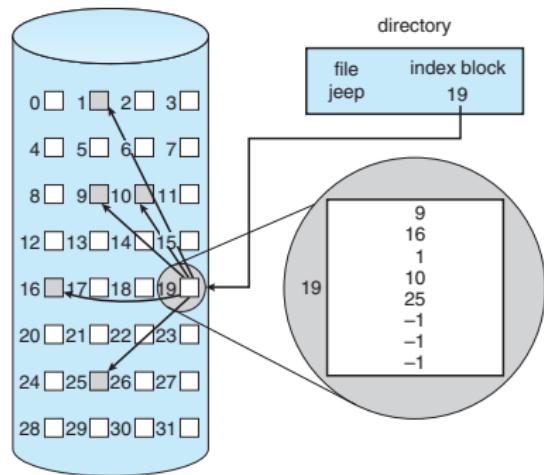
- **FAT: File Allocation Table**
- Tabla al inicio del disco con una entrada por archivo
- Último cluster almacena EOF (*end-of-file*)
- +Mejor acceso aleatorio
- -Limitado por tamaño de tabla (para FAT32, 4GB por archivo)



# Asignación de bloques

## Asignación indexada

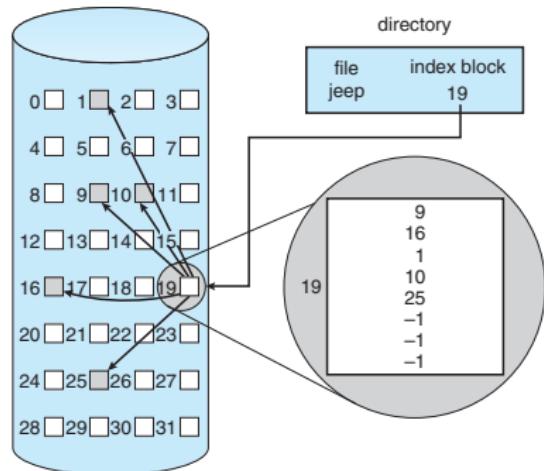
- Archivos contienen *index block*
- *Index Block* contiene bloques del archivo
- Último cluster almacena EOF (*end-of-file*)



# Asignación de bloques

## Asignación indexada

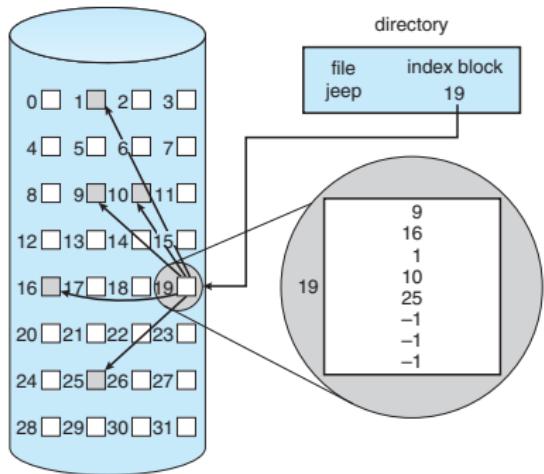
- Archivos contienen *index block*
- *Index Block* contiene bloques del archivo
- Último cluster almacena EOF (*end-of-file*)
- +Acceso aleatorio sin frag. externa



# Asignación de bloques

## Asignación indexada

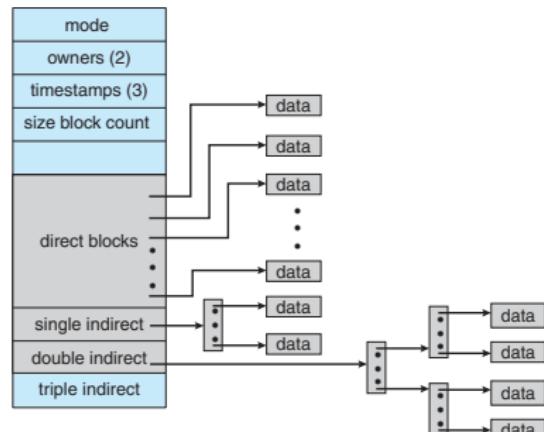
- Archivos contienen *index block*
- *Index Block* contiene bloques del archivo
- Último cluster almacena EOF (*end-of-file*)
- +Acceso aleatorio sin frag. externa
- -Se pierde espacio por tamaño de *index block*



# Asignación de bloques

Asignación indexada: tamaño del *index block*: NTFS, ext3, HFS

- **Esquema enlazado:** última entrada de *index block* apunta a otro *index block*
- **Índice multinivel:** similar a tablas de página multinivel
  - Ej: blocks de 4KB, permiten 1024 punteros (de 32-bit)
  - Dos niveles permiten direccionar 1048576 bloques → 4GB



- **Esquema combinado:** primeros  $p$  punteros son bloques directos, los siguientes apunta bloque de índice simple, los siguientes a bloque de índice doble, etc

# Finalmente...

Existe una multitud de sistemas operativos...



¿Cuál es mejor?

# Finalmente...

