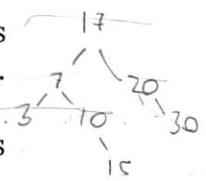




- Duración del examen: 3 horas.
- Todas las hojas entregadas deben tener nombre y DNI del alumno y Nº de página/Total de páginas.
- Las respuestas a los ejercicios deberán escribirse en pseudocódigo y estar claramente justificadas, es decir, acompañadas de una breve y clara explicación.
- En todas las preguntas pueden incluirse operaciones auxiliares, debidamente justificadas, si se considera necesario.

**(1.75 puntos) Ejercicio 1.-**

- Define que es un árbol balanceado o AVL y para qué sirve. **(0.25 puntos).**
- En un árbol binario de búsqueda, inicialmente vacío, indicar paso a paso las transformaciones realizadas al ir insertando los datos 17, 7, 10, 20, 15, 30, 3. Posteriormente, eliminar el 17. **(0.5 puntos).**
- En un árbol balanceado o AVL, inicialmente vacío, indicar paso a paso las transformaciones del árbol al ir insertando los datos del apartado b). Posteriormente, eliminar el 10. **(1 punto).**



**(3.25 puntos) Ejercicio 2.-** Se quiere trabajar con árboles binarios formados por elementos del tipo *objeto*. Suponiendo conocida la operación  $\_>=$ : *objeto* *objeto*  $\rightarrow$  bool:

- Dar la especificación del tipo abstracto de datos ARBOL\_BINARIO[OBJETO], deben especificarse claramente las operaciones básicas del TAD (tipos de entrada y salida de las mismas y ecuaciones de definitud) y describir su funcionamiento. **(0.25 puntos).**
- Ampliarla con las siguientes operaciones (pueden ser parciales):
  - b.1 altura*: árbol\_bin  $\rightarrow$  entero, calcula la altura de un árbol binario dado. **(1.0 punto).**
  - b.2 semicompleto*: árbol\_bin  $\rightarrow$  bool, comprueba si un árbol binario es semicompleto. **(1.0 punto).**
  - b.3 moticulo\_maximos*: árbol\_bin  $\rightarrow$  bool, comprueba si un árbol binario es un montículo de máximos. **(1.0 punto).**

**(1.75 puntos) Ejercicio 3.-** Se quiere trabajar con listas y pilas formadas por elementos de tipo *objeto*, pero del TAD *objeto* solo se conoce la operación  $\_==$ : *objeto* *objeto*  $\rightarrow$  bool que comprueba si dos objetos son iguales.

- Dar la especificación de los tipos abstractos de datos PILA[OBJETO] y LISTA[OBJETO], deben especificarse claramente las operaciones básicas del



TAD (tipos de entrada y salida de las mismas y ecuaciones de definitud) y describir su funcionamiento. **(0.50 puntos).**

- b) Ampliarlas con las siguientes operaciones (pueden ser parciales):

*b.1 lista\_union:* pila pila → lista, dadas dos pilas, obtener la lista unión de ambas. En la lista unión no deben aparecer objetos repetidos, el orden de los objetos no importa.

**(0.75 puntos).**

*b.2 lista\_intersección:* pila lista → lista, dadas una pila y una lista, obtener una lista con la intersección de ambas. En la lista intersección no deben aparecer objetos repetidos. **(0.50 puntos).**

**(3.25 puntos) Ejercicio 4.-** Se quiere trabajar con árboles generales formados por elementos de tipo *letra*, pero del TAD *letra* solo se conoce la operación `_==_`: letra letra → bool que comprueba si dos letras son iguales.

- a) Dar la especificación del tipo abstracto de datos ARBOL[LETRAS] deben especificarse claramente las operaciones básicas del TAD (tipos de entrada y salida de las mismas y ecuaciones de definitud) y describir su funcionamiento. **(0.25 puntos).**

- b) Suponiendo conocida la especificación del TAD LISTA2[LETRAS], escribir las siguientes operaciones:

1. *son\_iguales:* árbol árbol → bool, comprueba si dos árboles generales dados son iguales, tanto en forma como en contenido.

**(1.0 punto).**

2. *es\_imagen\_especular:* árbol árbol → bool, dados dos árboles generales de entrada, comprueba si el segundo árbol es la imagen specular del primero.

**(1.0 punto).**

3. *lista\_hojas:* árbol → lista, que genera una lista con todas las letras que se encuentran en los nodos del árbol de derecha a izquierda.

hoja

**(1.0 punto).**

## Pilas

Estructura lineal LIFO.

### Funciones

Generadoras

vacía  $\rightarrow$  pila

apilar: elemento, pila  $\rightarrow$  pila

Modificadoras

parcial desapilar: pila  $\rightarrow$  pila

Observadoras

parcial cima: pila  $\rightarrow$  elemento

vacía?: pila  $\rightarrow$  bool

Ejemplo 1: insertar 3, 5, 1 en una pila.

fun creapila () dev p:pila

p  $\leftarrow$  vacía

apilar(1, p) 

apilar(3, apilar(5, p))



$\Rightarrow$

$\Rightarrow$

ffun

Ejemplo 2: obtener suma de enteros de una pila

fun contar\_enteros(p:pila) dev n: natural

si vacía?(p):

dev 0

si no:

dev ~~cima(p)~~ + contar\_enteros(desapilar(p))

finsi

ffun

Ejemplo 3: invertir pila.

fun pila\_invertida(p:pila) dev (q:pila)

q  $\leftarrow$  pVacia()

mientras !vacía?(p):

apilar(cima(p), q)

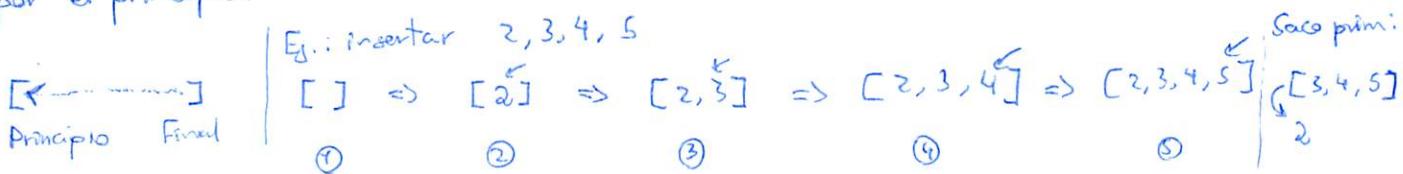
desapilar(p)

fmientras

ffun

## • Colas:

Estructura lineal FIFO. Se añaden elementos por el final y se sacan por el principio.



## • Operaciones:

### Generadoras

vacía? : cola → bool

círculo: elemento, cola → cola

encolar

### Modificadoras

parcial eliminar(cola) → cola  
desencolar

### Observadoras

vacía? : cola → bool

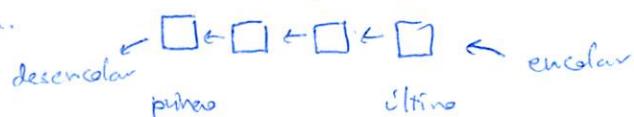
parcial primero: cola → elem

## Implementación

① Vectores: índice en el último elemento. Al borrar el primero, el resto se desplazan.

② Vectores circulares: se evita el desplazamiento.

③ Celdas enlazadas: cada celda contiene un elemento y un puntero a la siguiente celda.



Ej. 1: contar npares en colas concatenando la fun. espar(num) → dev. bool.

func npares(c: cola) dev: entero:

  n ← 0

  mientras !vacía?(c) hacer:

    | si: espar?(prímero):

    |   n ← n + 1

    |   eliminar(cola)

  fimientras

ffun

Ej. 2: concatenar 2 colas de forma recursiva.

func concatenar(c1, c2: cola) dev: c1: cola

  si: vacía?(c2):

  |   dev c1

  si: no:

  |   e ← desencolar(c2)

  |   dev concatenar(encolar(e, c1), c2)

  fse

ffun

Ej. 3: función que comprueba si 2 colas son iguales (usando == -)

func iguales?(c1, c2) dev: bool

  si: vacía?(c1) & vacía?(c2):

  |   dev T

  si: no:

  |   si: vacía?(c1) dev F

  |   si: vacía?(c2) dev F

  |   el1 ← desencolar(c1)

  |   el2 ← desencolar(c2)

  |   si: el1 == el2 entonces dev iguales?(c1, c2)

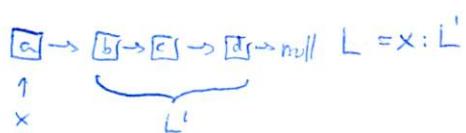
  |   si: no dev F

finales

## Listas

Conjunto de elementos ordenados linealmente de la forma cabesa: resto.

$x : L$   
↑      ↓  
cabesa    resto



## Operaciones

### Generadoras

$[] : \rightarrow \text{lista}$  (gen. lista vacía)  
 $- : \text{elem}, \text{lista} \rightarrow \text{lista}$  (add sig.)

### Modificadoras

parcial resto: lista  $\rightarrow$  lista ( $L'$ )  
parcial ult: lista  $\rightarrow$  lista (elim. ult.)

### Observadoras

vacia?: lista  $\rightarrow$  bool  
parcial ult: lista  $\rightarrow$  elem.  
parcial primero: lista  $\rightarrow$  elem.

## Implementación

### Celdas enlazadas:

primero  $\rightarrow$   $\square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null}$   
Add  $\textcircled{3} p \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null} // \textcircled{3}$   
 $\textcircled{3} p \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null}$

### Celdas doblemente enlazadas:

primero  $\rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null}$   
null  $\leftarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null}$

### Celdas doblemente enlazadas con acceso aleatorio:

primero  $\rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null}$   
null  $\leftarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \text{null}$

Array de punteros:

## Listas 2:

Añade las siguientes operaciones:

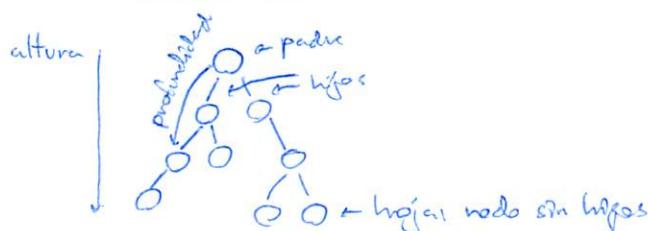
$[ - ] : \text{elemento} \rightarrow \text{lista}$  (lista unitaria)  
 $- \star - : \text{elem}, \text{lista} \rightarrow \text{lista}$  (add derecha)  
 $- + - : \text{listas}, \text{lista} \rightarrow \text{lista}$  (concatena listas)  
 $\text{long} : \text{lista} \rightarrow \text{natural}$  (dev. longitud lista)

### Listas ampliadas:

Añade las sig. operaciones:  
Lista[natural]  $\rightarrow$  elemento  
insertar (elem, lista, nat)  $\rightarrow$  lista  
modificar (elem, lista, nat)  $\rightarrow$  lista  
borrar (lista, natural)  $\rightarrow$  lista  
esta? (elem, lista)  $\rightarrow$  bool  
buscar (elem, lista)  $\rightarrow$  natural

## • Árboles binarios

Conjunto de nodos donde cada nodo posee un elemento y dos punteros a nodos hijos, llamados hijo izq. y derecho. El nodo del que welgan el resto se le llama nodo raiz.



## • Operaciones

### Generadoras

- $\Delta$ :  $\rightarrow$  arbol (ab. vacio)
- $\dots \rightarrow$  arbol, elem arbol  $\rightarrow$  arbol
- $\{$  (gen arbol a partir y con otros)

### Observadoras

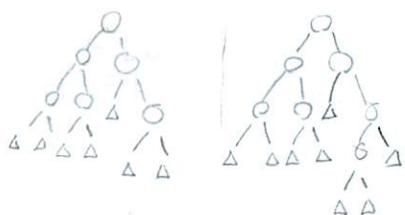
- |                                     |   |            |
|-------------------------------------|---|------------|
| raiz: arbol $\rightarrow$ elem      | } | parámetros |
| izq: arbol $\rightarrow$ arbol      |   |            |
| der: arbol $\rightarrow$ arbol      |   |            |
| altura: arbol $\rightarrow$ natural |   |            |
- vacio: arbol  $\rightarrow$  bool

### Ej. 1: Operación altura.

```
fun altura(ab: arbol) dev natural
    si vacio?(ab):
        dev error(Arbol vacio)
    si no:
        si (vacio?(izq))  $\wedge$  vacio?(der):
            dev 0
        fsi
        si (vacio?(izq))  $\wedge$  !vacio?(der):
            dev 1 + altura(izq(ab))
        fsi
        si (vacio?(izq))  $\wedge$  vacio?(der):
            dev 1 + altura(der(ab))
        fsi
        si (!vacio?(izq(ab))  $\wedge$  !vacio?(izq(ab))):
            dev max(altura(izq(ab)), altura(der(ab)))
        fsi
    ffin
```

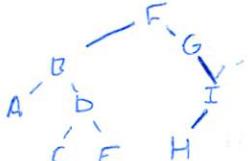
### Ej. 2: ver que ambos árboles tienen la misma forma.

```
fun iguales(ab1, ab2: arbol) dev bool
    si vacio?(ab1)  $\neq$  vacio?(ab2):
        dev F
    si vacio?(ab1)  $\wedge$  vacio?(ab2):
        dev T
    si no:
        dev iguales(izq(ab1), izq(ab2))  $\wedge$ 
            iguales(den(ab1), den(ab2))
```



## • Recorrido de un árbol

- Preorden: raiz, izq, der
- Inorden: izq, raiz, der
- Posorden: izq, der, raiz

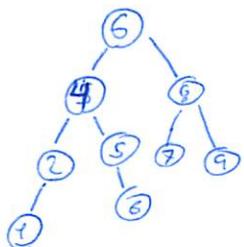


6as operaciones, devuelven listas con el recorrido.  
Ej: preorden(ab)  $\rightarrow$  lista  
 $L = [F, B, A, D, C, E, G, I, H]$

- Preorden: F, B, A, D, C, E, G, I, H
- Inorden: A, B, C, D, E, F, G, H, I
- Posorden: A, C, E, D, B, H, I, G, F

## • Árboles de búsqueda

Árbol binario donde los elem. del hijo izq. son menores o iguales que la raíz y los del hijo derecho son mayores que la raíz.



## • Insertar:

```

fun insertar(e: elemento, abb: abin):
  si vacío(abb) devolver ( $\Delta e \circ \Delta$ )
  si no:
    si  $e \leq (\text{raiz}(abb))$ :
      insertar(e, izq(abb))
    si no:
      insertar(e, der(abb))
  
```

## • Borrar:

① Borrar nodo sin hijos: se borra y el padre apunta a null.

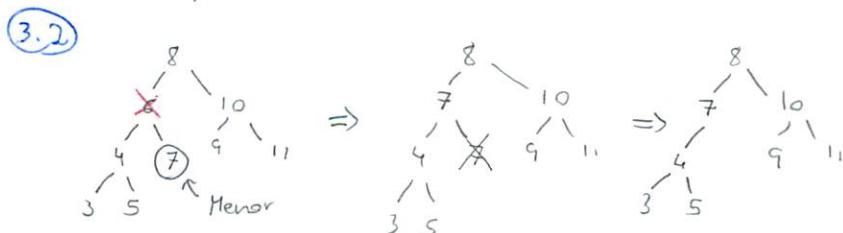
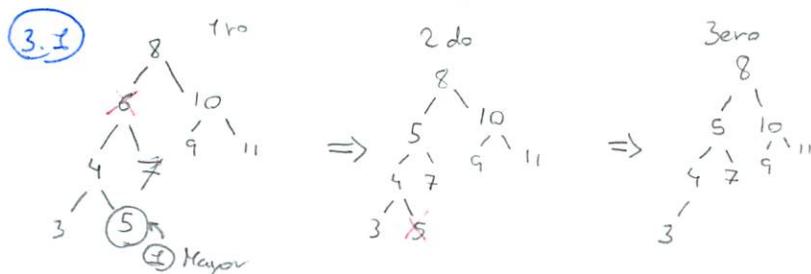


② Borrar nodo con un hijo: se borra y el hijo ocupa su lugar.



③ Borrar nodo con dos hijos: dos opciones:

- ① Buscar el mayor del hijo izquierdo
- ② Buscar el menor del hijo derecho



## o Árboles de búsqueda (2)

Borrar: pseudocódigo.

```
fun borrar (e: elemento, abb: arbol-busqueda)
    si !vacío(abb):
        si e < raiz(abb): } si es menor que la raiz buscar izq.
            borrar(e, izq(abb))
        si e > raiz(abb): } si no, buscar en la derecha
            borrar(e, der(abb))
        si e == raiz(abb): // Si es igual:
            si izq(abb) == null & der(abb) == null: } caso 1
                abb ← null // Borrado
            si izq(abb) != null & der(abb) == null: } Caso 2
                abb ← izq(abb)
            si izq(abb) == null & der(abb) != null:
                abb ← der(abb)
            si izq(abb) != null & der(abb) != null:
                maxi ← maximo(abb.izq)
                borrar(maxi, abb.izq)
                abb ← (izq(abb) - maxi - der(abb)) } Caso 3.x
            fsi
        fsi
    fin
```

## • Árbol K-Ario (Árbol general)

- Árbol de grado k. Cada nodo puede tener de 0 a k hijos.

↑  
nº máx  
de hijos  
que puede  
tener un  
nodo.

↑  
A estos m hijos  
(donde  $0 \leq m \leq k$ )  
se les llama hijos.

- Full    - Árbol homogéneo: todos los nodos tienen k hijos. (Excepto hojas).
- Perfect    - Árbol completo: árbol homogéneo con hojas a misma profundidad
- Complete    - Árbol casi completo: árbol completo al que se eliminan 0 o más hojas empesando por la derecha

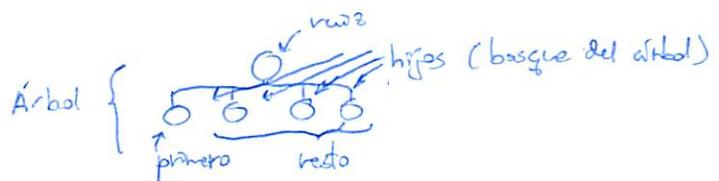
## • Operaciones

### Generadoras

- o : elemento basque  $\rightarrow$  Árbol\_gen.
- [ ] : basque (basque vacío)
- \_ : árbol basque (árbol basque)

### Observadoras

- raiz (árbol)  $\rightarrow$  elem
- hijos (árbol)  $\rightarrow$  basque
- vacio? (basque)  $\rightarrow$  bool
- long? (basque)  $\rightarrow$  natural
- num\_hijos (árbol)  $\rightarrow$  natural
- primeros (basque)  $\rightarrow$  árbol
- resto (basque)  $\rightarrow$  basque (dev. el basque son el resto del árbol)
- subárbol (árbol, natural)  $\rightarrow$  árbol (acceso al k-ésimo hijo del árbol)



## ○ Árboles AVL

Árbol binario de búsqueda balanceado. La altura del hijo der es igual. Se define en I.

Factor de balanceo = altura der - altura izq.

Cada nodo tiene:

Permite las operaciones típicas de abb.

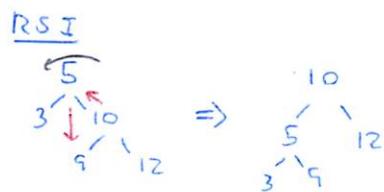
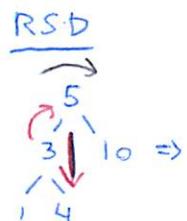
- Raiz
- Altura
- Izq.
- der

Balanceo: 4 operaciones

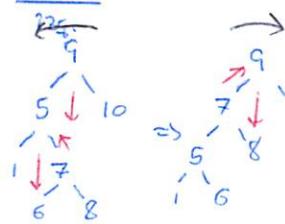
{ Rotación simple { Izq.  
der }

{ Rotación doble { Izq (der, Izq)  
der (Izq, der) }

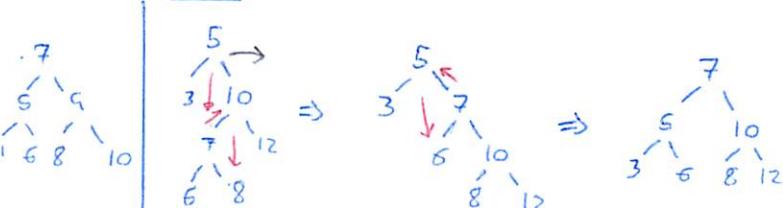
Ejemplos:



RDD



RDII



## • Montículos (Heap)

Completa 2 propiedades:

- Es un árbol binario completo: todos los nodos completos salvo quizás el último, donde todas sus hojas están lo más a la izquierda posible.

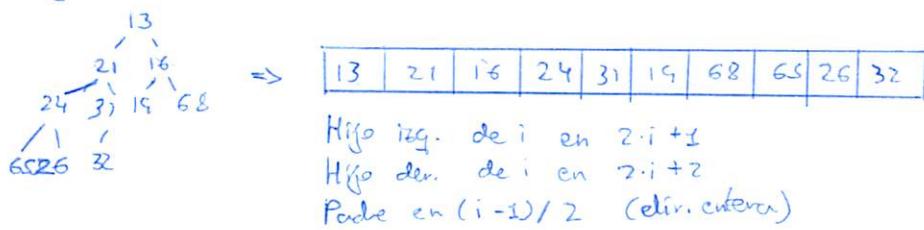


- Orden: para cada padre, sus hijos son mayores o iguales que el padre.



Un heap o montículo se puede implementar como un array: leemos el ab. por niveles.

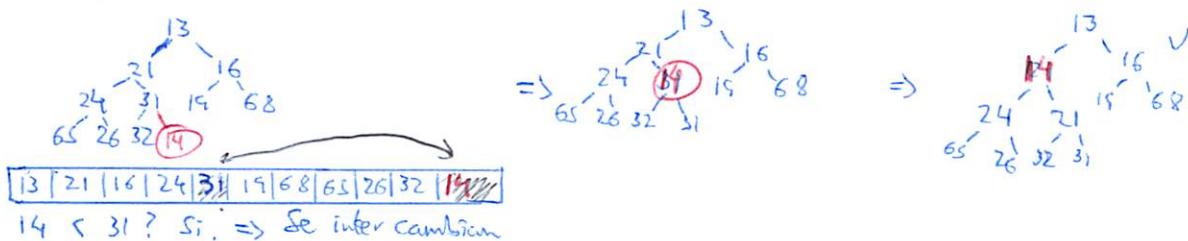
Q:



Insertar: se inserta en el primer bloque libre y se le hace flotar hasta que mantenga la coherencia con el montículo.

• Reflotar: se compara el padre con el hijo. Si hijo < padre, se intercambian.

Ejemplo: insertar 14.



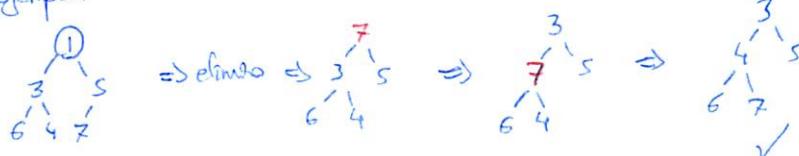
Borrar: siempre es el primer elemento. Se sustituye la pos 0 por la última, y se hunde el dato hasta su pos. correcta.

$$\boxed{1 | 2 | 3 | 4 | 5} \Rightarrow \text{eliminar } 1. \Rightarrow \boxed{5 | 2 | 3 | 4} \Rightarrow \text{Hundir.}$$

• Hundir: se hunde en la der. del hijo menor.



Ejemplo:



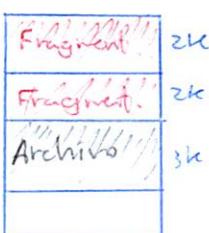
## SS OO Av.

### • Fragmentación

Desaprovechamiento de memoria



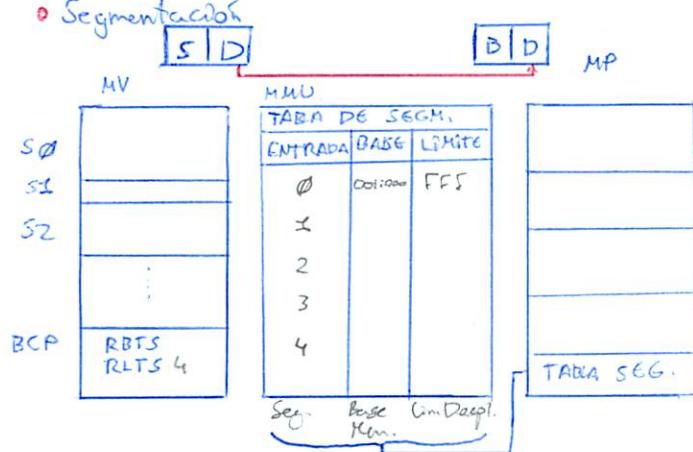
Internos



### • Espacio dir.

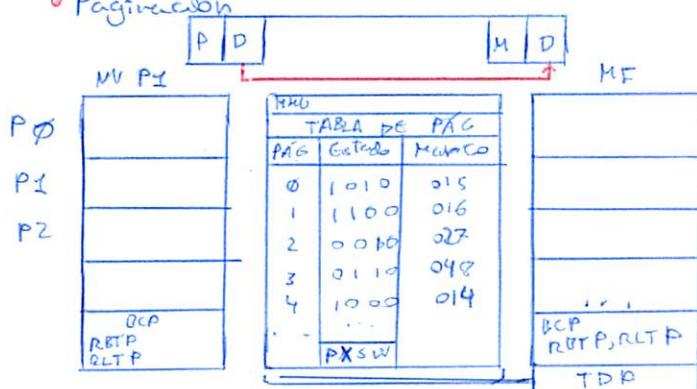
La MMU traduce dir. virtuales a dir. físicas  
 HV: índice para cada proceso  
 MP: Compartido por procesos.

### • Segmentación



• Segmentos de tam. variable

### • Páginaación



• Páginas de tam fijo.

• Presencia

✗ → efectable

S → Compartido

W → Escritura.

## Ejercicio 2 - Sist. Archivos UNIX

$$\text{Max no bytes} = 2^{32}$$

$$\text{Panino blagie} = 1024$$

Super bloque: info del ferrocarril

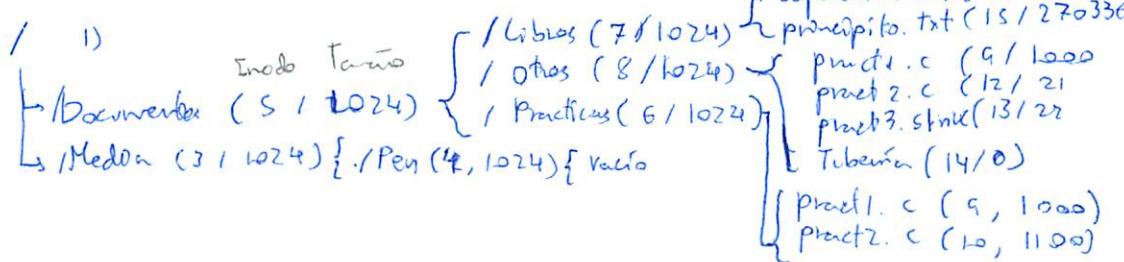
Cada modo ocupa 32 B y almacena  
DEG tipo LNK (comb(2)), Tubería FIFO.

- Tipos de archivos: REG, DIR, LNK (símbolico), Tuberia
  - Guardar LNK
  - Tam. archivo en bytes
  - 2 punteros dir. Ptr1 Ptr2
  - 2 pts. indir. simples Ptr Ind Sim1 Ptr Ind Sim2
  - 2 pts. indir. dobles Ptr Ind Dob1 Ptr Ind Dob2

Example

INODE	2	
2	DIR	5/1024150
		NULL...
-	-	-
27	REC	1 800

→ Include bogue files  
S1208) introduced



## Super Blogue :

Tan blogue dulos: 1024

Nº blagues dans: 60000

Nº nodos indice: 60000

Mapas de bits. (Tableaux d'aspects)

Tabla nodo indice: {y bloques}

Tabla nodos indice: papas a nodo en bloques		2	3	4	5	6	..	9	10	11	12	13	14	15
Nodo		Dir	Dir	Dir	Dir	Dir		REG	REG	REG	SINK	LNK	FIFO	REG
Tipo														
Ent.	4	3	1	4				2	2	1		1	1	
Tam(B)	1024	1024	1024	1024	1024	1024		1000	1100	5120	21	22	0	
PDif1	0	1	2	3	4			7	8	10	15	16	Null	17
Pdif2	N	N	N	N				NULL	9	11	NULL	NULL		18
PIAdS1	0	0	0	0				NULL	12	1				19
PIAdS2	0	0	0	0				0	0	0				270
PIndDI	0	0	0	0				0	0	0				
PIndDA	0	0						0	0	0				

## Table bloque de datos

## T4.- Sist. Op. Av. - Sist. Archivos

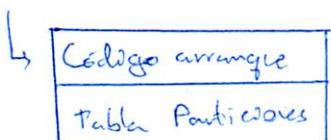
f disk /dev/hda

stat archive.

enl. simbólico ln -s orig. destino  
enlace fuerte ln origen destino

Partitionado de disco:

MBR | Boot 1 | Part. | Boot 2 | Contenido | Boot 3 | Contenido | Boot 4 | Part. Cdg. 1 # Part. Us. 2 |

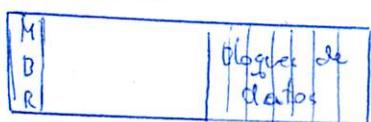


### Ej. Problema I

Est. Tabla partición

00 Estado (1B)	80 → (1000 0000) → Part. activa
01 Inicio (3B)	0001 01
04 Tipo (1B)	07 → NTFS
05 (3B)	FFFF FE
08 Dist. (	00 00 00 3F
0C N° Sector (4B)	07 80 1F 1A

- Formateo: crear y instanciar variables y tablas de un sistema de archivos.  
/dev/hda3 → Sist. Archivos de UNIX.



Est. Datos con  
info. sobre  
Cmo se  
Guardan los  
datos

Un archivo en UNIX se identifica por su Inodo:

Inodo

- 12 punt. directos → a datos
- 1 punt. indirecto → a puntero(a Inodo)
- 1 punt. doble → a Inodo → a Inodo
- 1 punt. triple. → a Inodo → a Inodo → a Inodo

Enlace fuerte: <sup>in=5</sup> ./practica1 N/practica1\_enlace1

Enlace simbólico: ln ./practica1 ./practica1\_slink

Lab 30% / PGC's 30%  
C.Final 40%

- ④ SO. Avanzados

Prog. C

Info que estructural

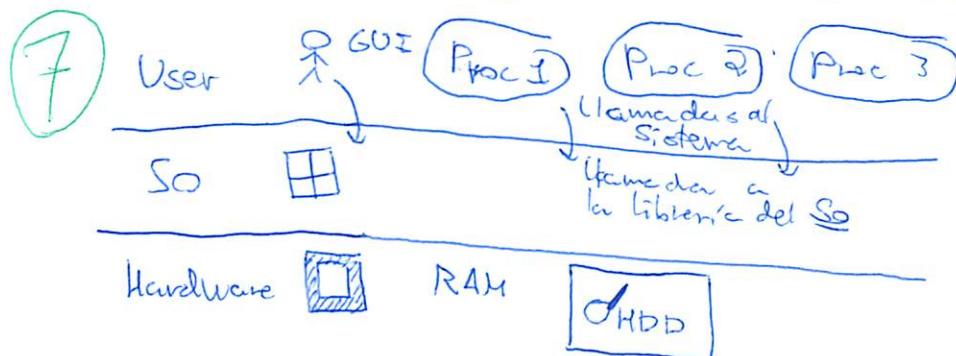
→ Gest. Memoria  
→ Gest. I/O  
→ Gest. archivos

- Personas  
- Procesos

↳ Preguntar a la mínima

Def Formal: software que pone los recursos físicos y lógicos a disposición del usuario de forma segura y eficiente

SO: herramientas que comunican al usuario con el hardware



6 Ejemplo:

Ejemplo 2 cat notasenero2018.txt → Pasa por capas → CPU → Busca parte HDD. → la muestra

1 ¿Cómo gestiona el SO las solicitudes de usuario?

↳ Llamadas al sistema (ej. 1)  
(Funciones de biblioteca que realizan llamadas al sist.)

2 o Comunicación user - SO ⇒ Func biblioteca que realizan llamadas al sistema

3 o Llamadas al sistema: servicios que tienen al SO.

• Prácticas

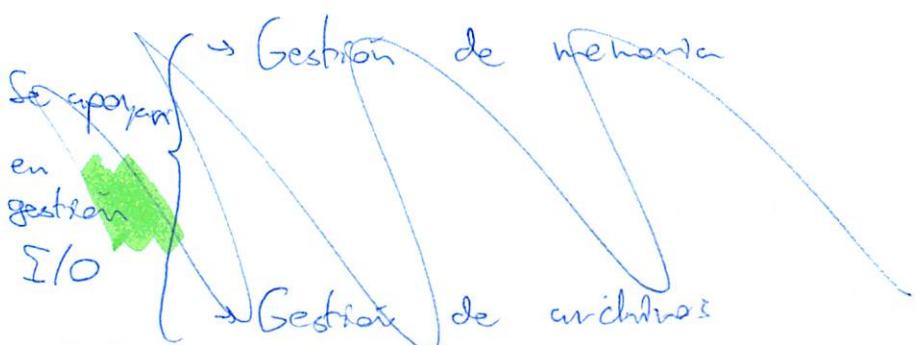
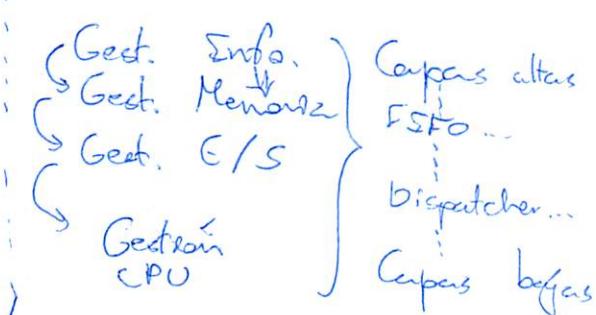
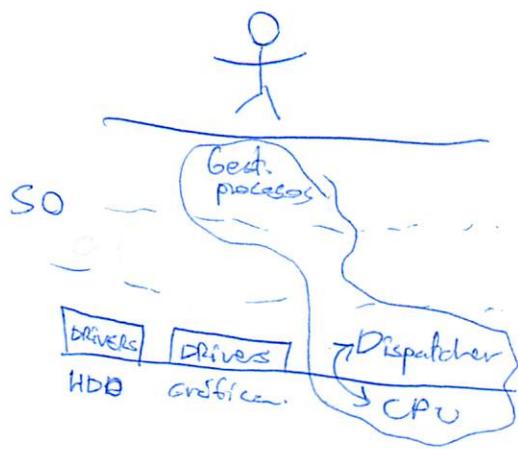
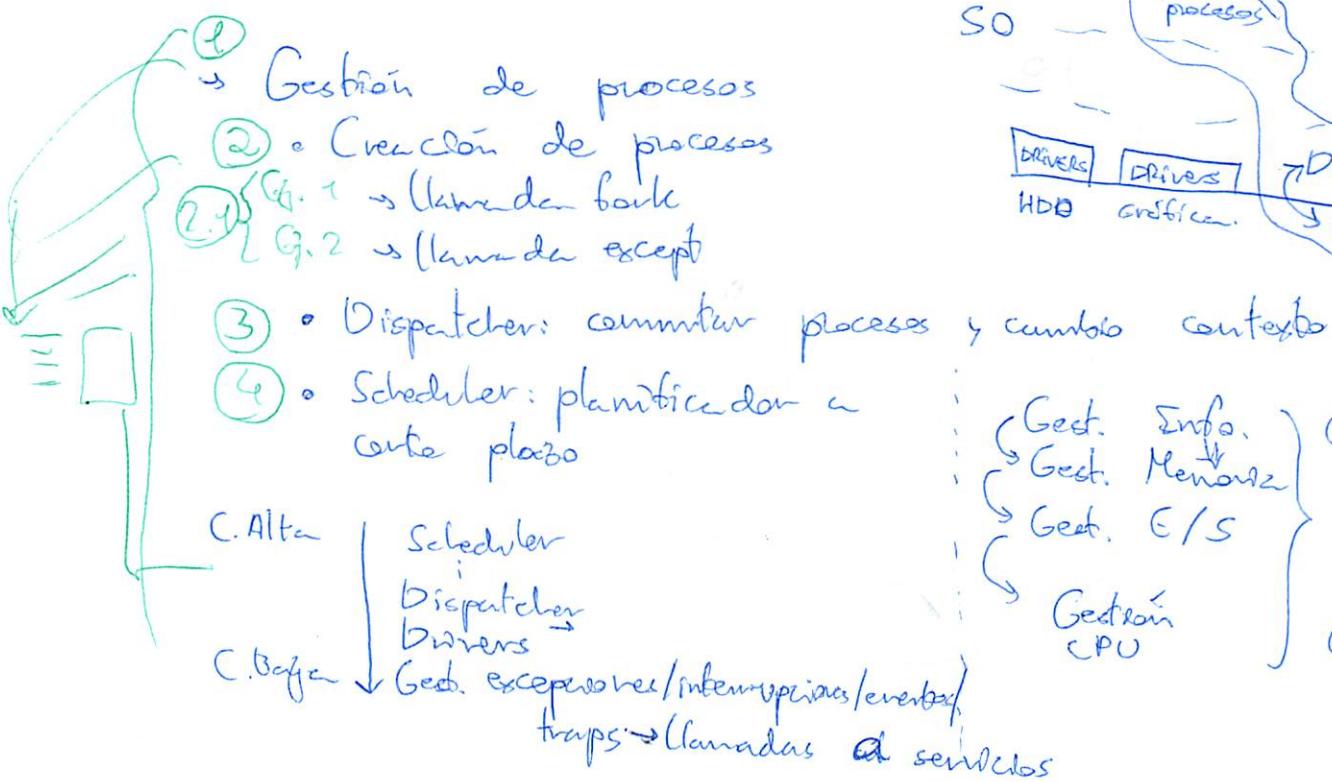
1 → Comprensión

2 → Pseudo código

3 → Programar

Falta el 7 de diciembre y nosque otra dva

## Capas del SO



### Diferentes diseños:

Núcleo	Kernel + {	Ejecutado en modo supervisor/protegido
Unix	Kernel + {	modo supervisor/protegido
VxWorks	Solo Kernel {	modo proteg.: dispatcher, etc. → Muy bajo nivel Diseño microscópico

Diseño micronúcleo:

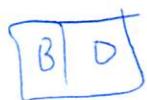
Diseño monolítico:

## 2 Segmentación:

- Mecanismo de la gestión de memoria para acceder a ella.
- Prod. freg. externa.
- Segmentos dinámicos.

4

Qj. Segmentación



S0  
S1  
S2  
S3  
S4

	0000
	FFFF
S0	0000
	0000
S1	0600
	0000
S2	0700
	0000
S3	0000
	FFFF
S4	0000
	0000

T. Segm.

S	B	L
0	0000	0000
1	1	06
2	1	1
3		
4	1	1

4

	0000: F680
S0	0000: FF80
S1	0300: 0000
S2	0300: FFFF
S3	0301: 0000
S4	0301: C000
S5	0600: 2000
S6	0600: 2700
S7	0F00: 0000
S8	0F00: FFFF

Marcos  
Marcos  
Marcos  
Marcos  
Marcos

## Ej. 5500 - Segmentación

4) Formato de acceso de segmento:

Contenzo seg. 1. Tamaño (Límite)

Dirección Desplazamiento

32 bits = 4 bytes

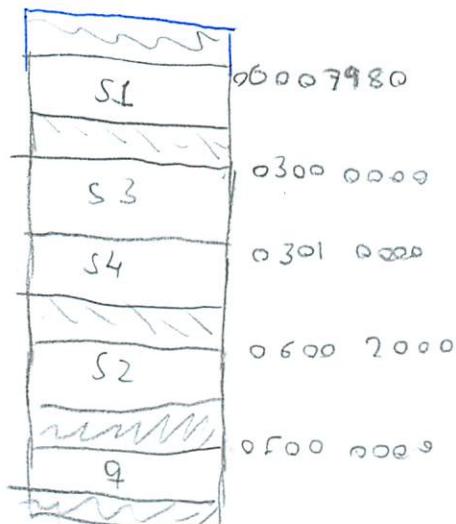
Control de Segmentación:

Ejercicio 4.- Operación 01

S	B	Lím.
5	0F00:0000	FFFF
4	0000:7980	0600
3	0600:2000	0700
2	0300:0000	FFF F
1	0301:0000	C000

S2: Empieza 0600:2000 Acaba 0600:2790

①



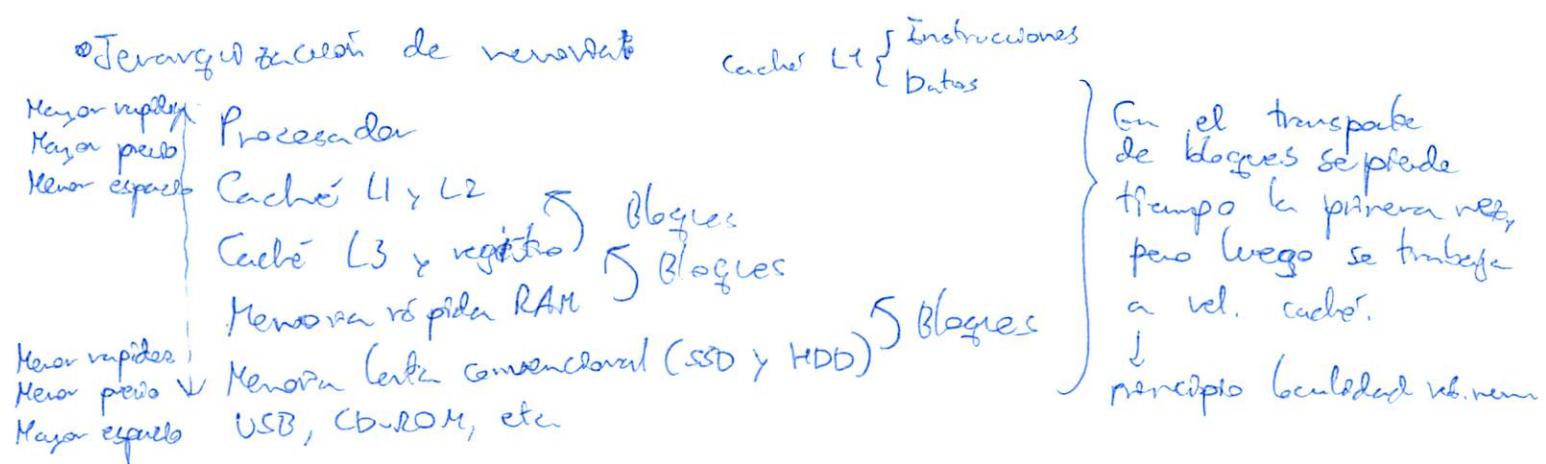
Seg. Despl.

- ③
- 0004 0202  $\Rightarrow$  0301:0202;
  - 0004 0898  $\Rightarrow$  0898 > 0000; No es posible, se pasa del límite.
  - 0003 000A  $\Rightarrow$  0300:000A; ✓
  - 0000 0509  $\Rightarrow$  0F00:0509; ✓
  - 0001 06FF  $\Rightarrow$  06FF > 0600; Se pasa del límite.
  - 0002 0701  $\Rightarrow$  0701 > 0700; Se pasa del límite.
  - 0005 0007  $\Rightarrow$  Wtf. No hay seg. 5.

RBT S: Registro base de la tabla de segmentos: Indica donde empieza el reg.

RLTS: Registro límite de la tabla de segmentos: Indican máx. de segmentos

- Espacio de direcciones de un proceso: (Parte de cada proceso)
  - Conjunto de direcciones referenciables
  - Los procesos solo referencian direcciones virtuales
  - Se necesita un traductor de direcciones virtuales a dirección real
  - La MMU: gestiona traducción de memoria virtual a física



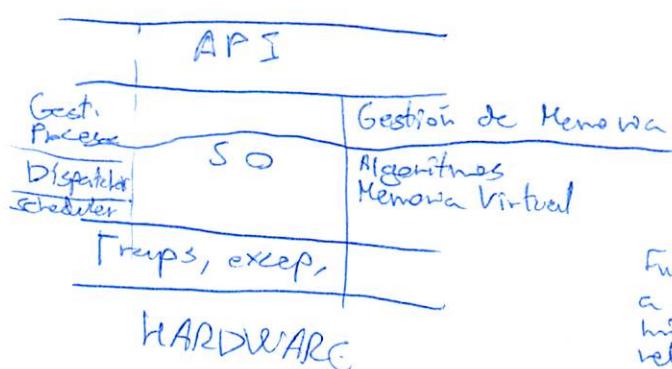
### • Principio Localidad Ref. Memoria

Los procesos tienden a concentrar sus referencias en un intervalo de tiempo en un subconjunto de su espacio de direcciones.

(Una pequeña parte del código se lleva todo el tiempo de procesamiento)

- Localidad espacial: una vez leída una referencia, es probable que las localidades cercanas sean también referenciadas. (Se crean de forma contigua)
- Localidad temporal: una vez hecha ref. en posición de memoria, es probable que esa posición vuelva a ser accedida en un instante cercano.

## SO Atañendos - Clase 2



## Dif. programación de procesos

### Memoria Virtual

Permite páginas en el HDD

Ventajas:

- Puedo correr muchas apps.

Inconvenientes:

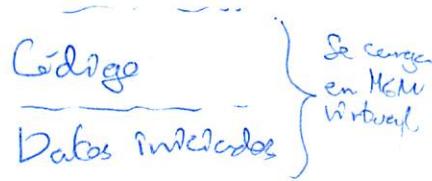
- Velocidad del HDD y del SSD
- Si se peta la RAM y no se pide despaginar a suficiente velocidad se jalan el invento.

### Archivos ejecutables

Número Mágico → permite al sistema saber el formato y ejecutable.

se cargan en memoria + CP Inicial

#### Tabla secciones



los procesos que vienen de p. ejecutables se cargan en la memoria principal en un espacio de direcciones virtuales en bloques independientes.

### Tabla de símbolos

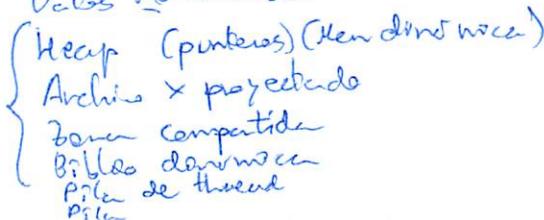
#### ↓ Mapa de Memoria

Códigos

Datos iniciados

Datos No iniciados

creadas en tiempo de ejecución



## G. Memoria

- ⇒ los procesos se ejecutan Solo en la memoria principal (y en paralelo)
- ⇒ Programa real: procesador y memoria principal

### Diseño procesador

→ Core(s)

Funciona a la misma velocidad

{ → Caché L1 } cada núcleo tiene L1, L2

{ → Cache L2 }

{ → Caché L3 } memoria compartida por cores

## ② Class 2

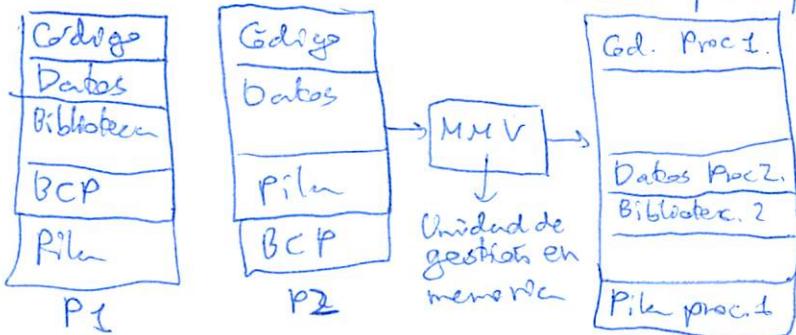
Mem. initial: especie Indep. para cada proceso

Memo principal: espacio compartido por todos los procesos en ejecución

Localidad espacial: se hacen las referencias a bloques de programa cercanos entre si.

temperatura; dado un hogar llamado  $s$  ejecutado en tiempo  $t$ , es probable que vuelva a ser llamado  $s$  ejecutado en  $t + \Delta t$ .

Mem. Virtual      Mem. physical



La primera vez que se pide la ejecución de un proceso es muy lenta, porque:

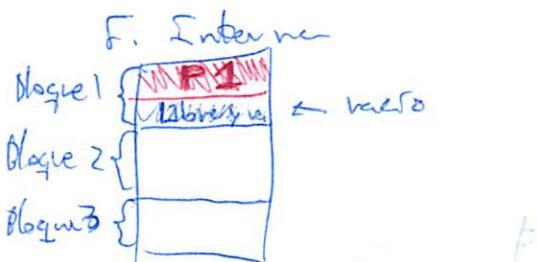
- Hay que cargar segmentos de la memoria virtual a la principal
  - Deben traducirse las direcciones de mem virtual a la principal.

## Fragmentación

fragmentación Desaprovechamiento de memoria libre disponible debido al mecanismo de memoria utilizada.

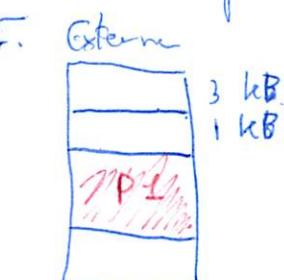
Fragmentación externa: entre particiones. Debido a que los segmentos son de diferente tamaño. Solución: compactar memoria

Fragueta del interior descubriendo el espécimen del siguiente:



Se produce cuando  
se trabaja con  
blogues tem. tipo.

No tiene solucion PERO  
tiene mas resultados.



Se producen bloques  
tamaño variable.

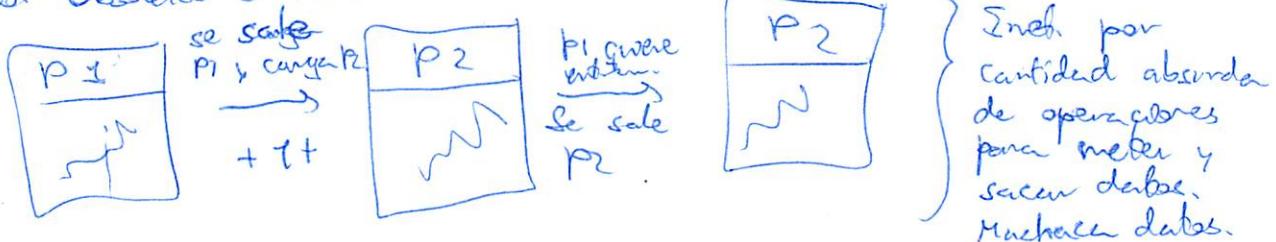
Tiene solución PERA  
requiere compactación y  
requiere procesamiento para  
ello.

## Reubicación

Proceso de asignar direcciones a los diferentes partes del programa.

- e) Reubicación dinámica: dirección virtual a real en tiempo de ejecución.  
- Necesita HW adicional: MMU.

- e) Reubicación estática: los procesos se cargan siempre en el mismo espacio. Obsoleta e ineficiente.



## Segmentación

### 4) Ciclo 2016

Esp. virtual: 4 GB por proceso.

Esp. Mem Principal: 4 GB para todos los procesos.

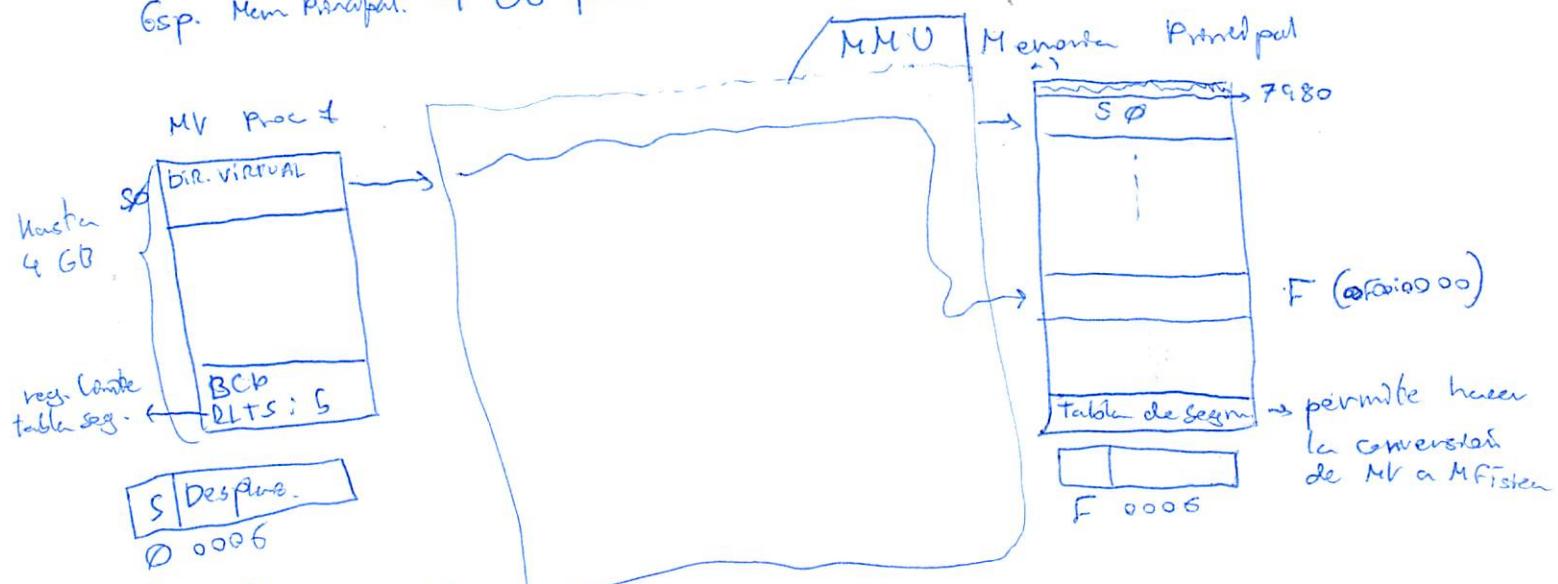


Tabla segmentos

S.	Base	Límite
0	0F 000000	FFFF
1	0000:7980	0600
2	0600:2000	0700
3	0300:0000	FFFF
4	0301:0000	0000

b) La tabla de segmentos

c) Mem. Virtual  $\rightarrow$  Mem. principal:  $\rightarrow$

$$a) 0x0004 0202 \Rightarrow S4: 0301 0000 + Despl. = 0301 0202$$

$$b) 0x0004 b898 \Rightarrow S4: 0301 b898 \text{ [Error! Límite < 0000]}$$

Quiero acceder a S3 d 123  
 $S3 \quad 0300 \quad 0000$   
 $+ Despl. \Rightarrow 0300 \quad 0000$   
 $+ \quad \quad \quad \quad \quad 123$   
 $0300 \quad 0123$

e) 06FF  $\rightarrow$  Lm. S4  $\rightarrow$  Error

f) 0002 2701

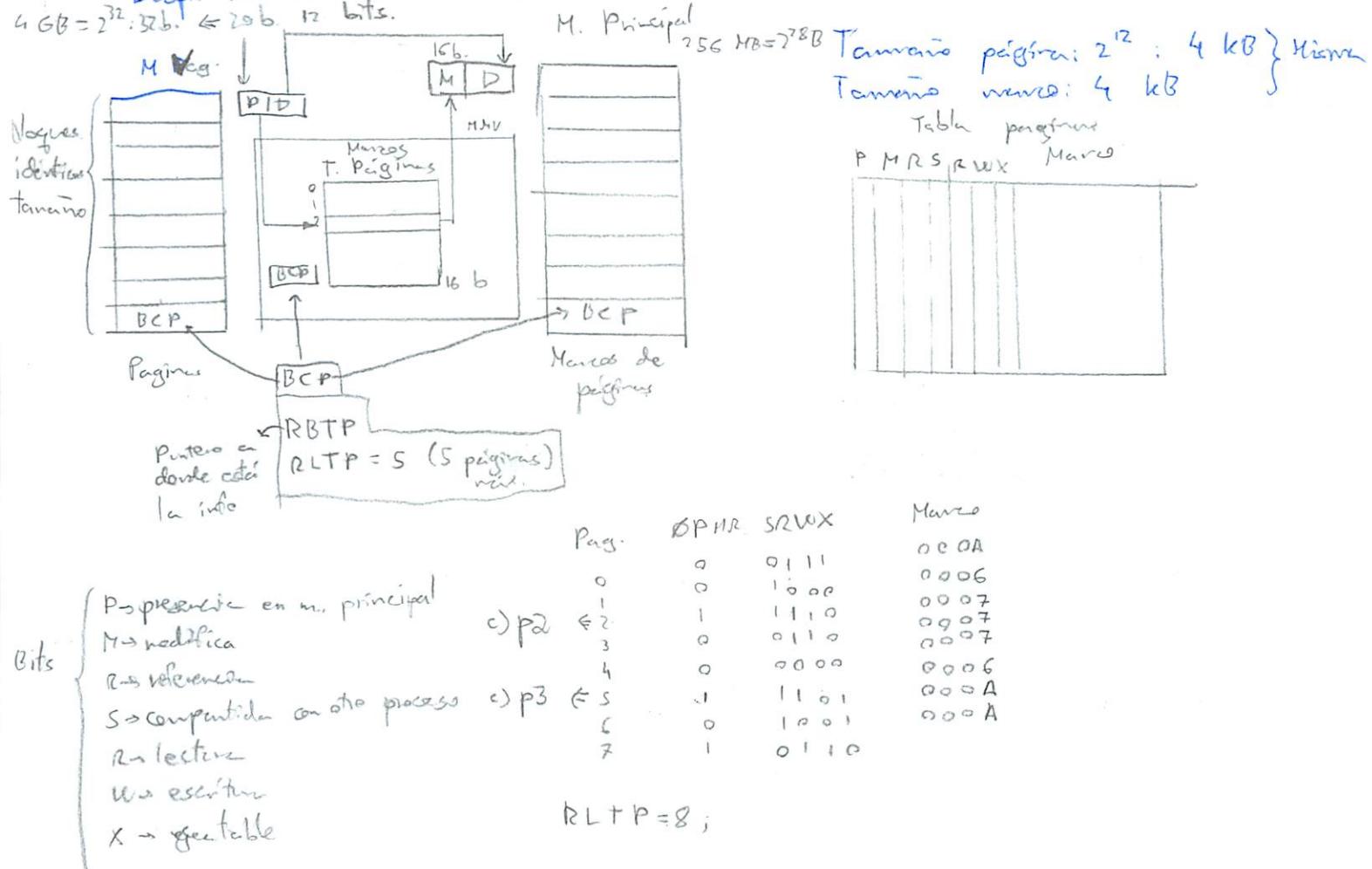
0701  $\rightarrow$  Lm. S2

## Paginación

- 3 Capacidad máx. direcc. virtual: 4 GB, para cada proceso. ( $2^{32}$  B)  
 Capacidad máx. direcc. física: 256 MB, para cada proceso. ( $2^{28}$  B)

Despl. 12 bits; Bty 3 bytes (000 → FFF)

$$4 \text{ GB} = 2^{32} \cdot 32 \text{ b.} \Leftrightarrow 20 \text{ b. } 12 \text{ bits.}$$



Pág.	0 P H R	S R W X	Marco
0	0	0 1 1 1	000A
1	0	1 0 0 0	0006
2	1	1 1 1 0	0007
3	0	0 1 1 0	0907
4	0	0 0 0 0	0007
5	1	1 1 0 1	000A
6	0	1 0 0 1	000A
7	1	0 1 1 0	000A

$$RLTP = 8;$$

### b) Traducción:

	Marco	Despl.
0000 0202	000A	202
0000 2898	0007	898
0000 500A	0006	00A
0000 7509	000A	509
0000 66FF	000A	6FF
0000 8701	0008	701

Página despl.  
20 b 12 b

### Excepción

Pero como  $P=0$  no está en memoria, el sistema busca un bloque libre  
 ✓ Todo correcto pg.  $P=1$   
 ✓ "  
 X-  $r=0$  !  
 X No existe; límite excedido. Excepción!  $P8 > RLTP$

- c) P2 comparte pag. 2 y pag 3 con P1. ¿Tabla de páginas para entidades 2 y 3 del P2?

R: hay procesos que comparten páginas. En m. virtual NO la comparten, pero en m. principal SI.

	TP	datos	Código
2	0100	1110	0007
3	0100	1101	0006

Mecanismo copartición: se ref. mismo bloque en mp. para tabla pag/seg.

Mecanismo protección: depende de la gestión de excepciones.

## • Sistemas Operativos Avanzados

- Profesora: Elena Campo (elena.campo@uh.es)

• Se seguirá un enfoque estructural en la asignatura:

⇒ Gestión de memoria } Apoyados en gestión I/O  
⇒ Gestión de archivos }

- Evaluación: P

⇒ Procs: 30 %.

⇒ Laboratorio: 30 %.

⇒ Examen final: 40 %.

## • Repaso de 1ro:

- Procesos ...

- Persona ...



⇒ SO: software que pone a disposición del usuario los recursos físicos y lógicos del hardware de forma segura y eficiente.

⇒ SO: herramienta(s) que comunican al usuario con el hardware.

## • ¿Cómo gestiona el sistema operativo las solicitudes de usuario?

Mediante:

### • Llamadas al sistema:

→ Mecanismo que utiliza un proceso para solicitar un servicio al SO, mediante bibliotecas del sistema.

### • Estas bibliotecas:

• Relacionan los procesos con el SO.

• Gestiona detalles de bajo nivel para transferir info al Kernel.

• Gestiona comutación a modo supervisor.

### • Ejemplo 1:

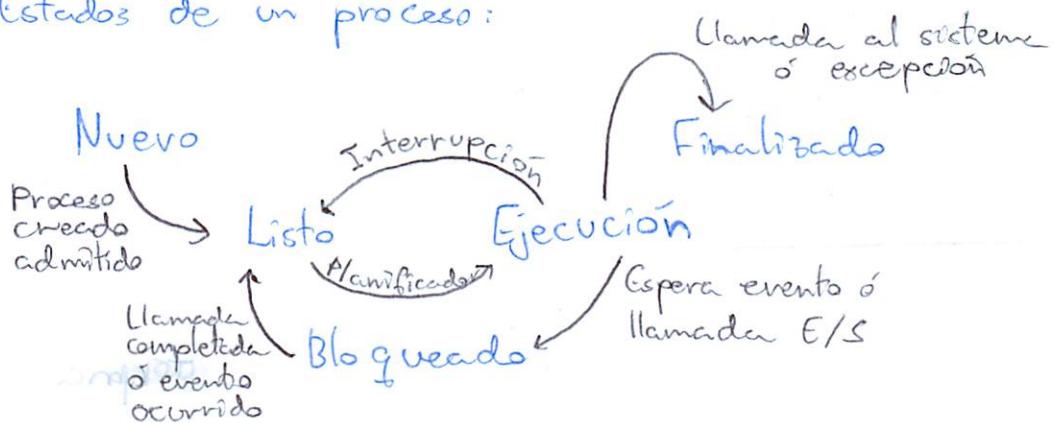
cat notas.txt → Pasa por capas → CPU → Busca parte HDD → print. (echo)

### • Ejemplo 2:

Open } Funciones de  
Read } biblioteca que  
Write } realizan llamadas al sistema

## • Capas del SO

- Gestión de procesos:
    - Kernel: proporciona acceso seguro a bajo nivel a los recursos.
    - Creación de procesos: llamada fork, except..
- ⇒ Def. proceso: programa en ejecución.
- ⇒ Estados de un proceso:



- Dispatcher: conmuta procesos y cambio de contexto.
- Scheduler: planificador a corto plazo de procesos.

User	R	Proceso 1	Proceso 2	
S O		Gestion de procesos Gestion Info Gestion Menú	Llamadas a bibliotecas Llamadas al sistema	Capas altas Algoritmos Dispatcher
Gestor I/O		Drivers	Dispatcher	Capas bajas
Hardware		HDD	Procesador	RAM

Diferentes diseños:

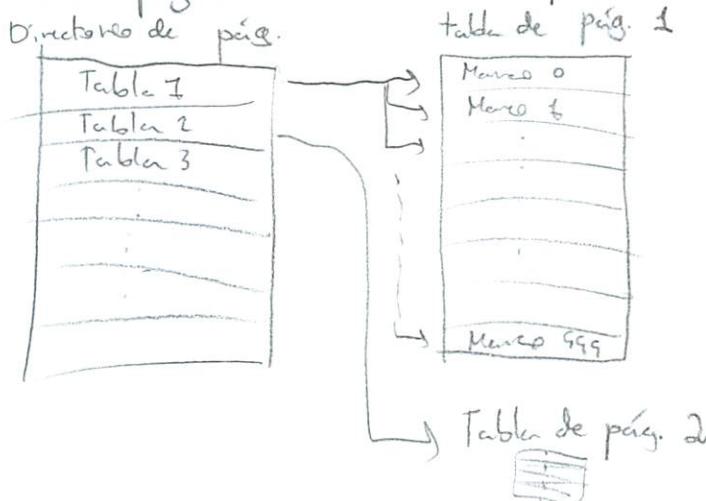
- Mononúcleo/Monolítico: kernel compilado y ejecutándose en modo supervisor. Llamadas al sistema. (UNIX)
- Micronúcleo: añade una capa de abstracción: servicios.

## • Repaso Tf - 5500 Av.

### • Segmentación paginada

- Tabla de páginas ext. MP  $\Rightarrow$  Queda mucho espacio para prog. con muchas entradas.

↳ Solución: páginar la tabla de páginas.

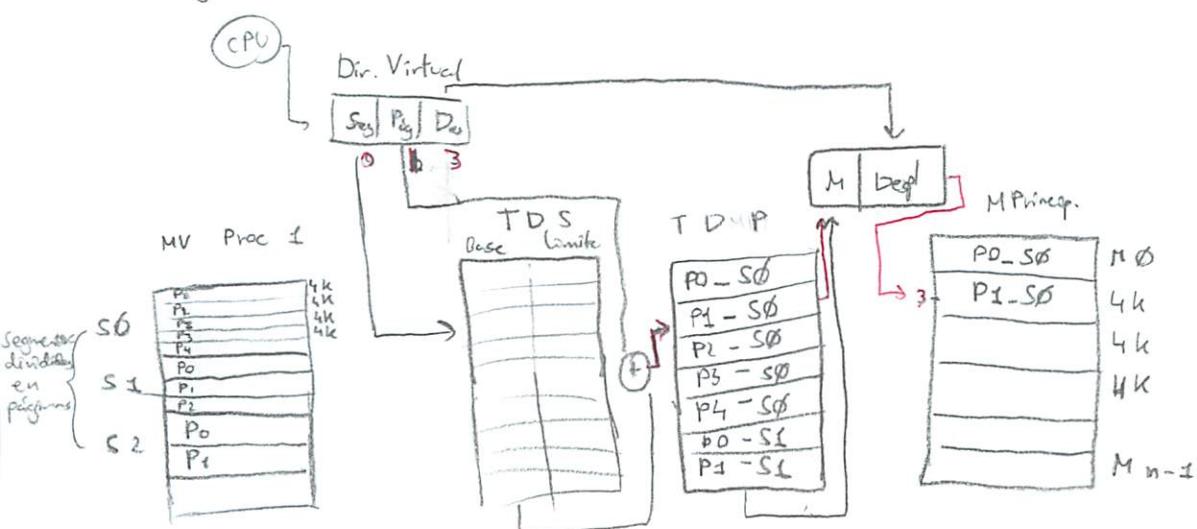


- Inconvenientes: más accesos a memoria, más lentes.

- Ventajas: por el principio de localidad de ref. esto funciona bien.

### • Segmentación paginada

Ejercicios:



## Ej. Examen Segmentación paginada

Seg.: 4 bits

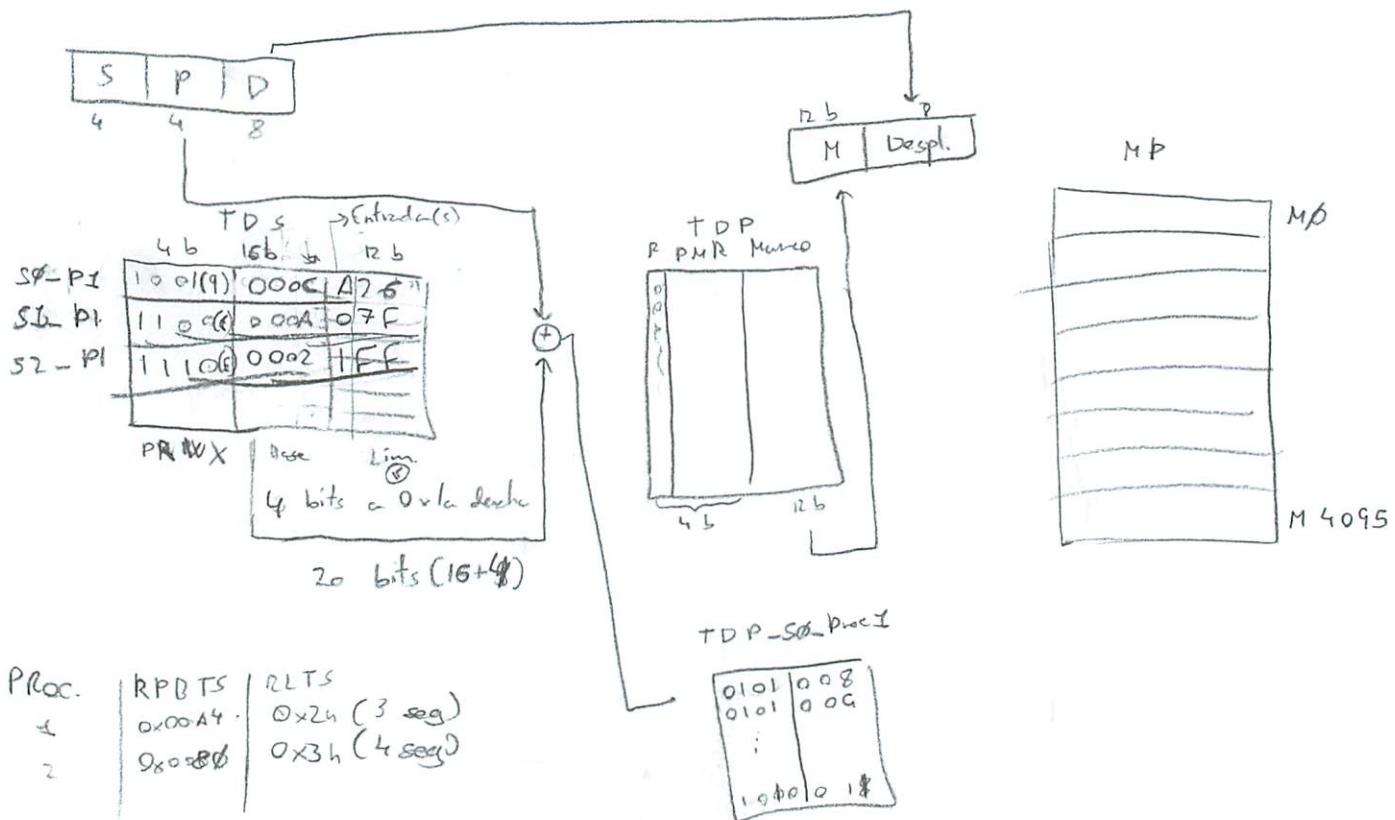
Despl.: 8 bits

Pág.: 4 bits

MP:  $2^{12}$  marcos

Marcos:  $2^8 = 256$  bytes

Tamaño MP:  $2^8 \cdot 2^{12} = 2^{20} = 1 \text{ MB}$



TS P1			TS P2		
PRW/X	Base	Limite	PRW/X	Base	Limite
(A) 1001	000C	A26	(A) 1001	000C	A26
(C) 1100	000A	07F	(C) 1001	000A	FFF
(E) 1110	0002	FFF	(C) 1001	0006	3FF
			(C) 1000	0004	07F

## T 2 - SO : Memoria Virtual

- Permite desplazamiento
- Almacenamiento secundario comp. a la MP
- Basado en localidad de ref.
- Se accede a mem. virtual siempre.
  - ↳ El tam. de los procesos depende del espacio de mem. virtual.
- Aumenta el grado de multiprogramación (más procesos simultáneamente en MP)
- Reduce la I/O  $\Rightarrow$  sólo se carga en MP lo necesario.

### ○ Requisitos

⇒ Los de la MMU:

- Página y segmentación

Tranf. más simples con bloques de tamaño fijo.

- Hardware de paginación: Incoherencia entre MV y MP

- Tabla de páginas

- Bits de pres., modificación, referencia

- Almac. auxiliar (HDD) (MV)

- Soporte para interrumpir instrucciones.

Ej.: al ejecutar una instrucción que ocupa más de una página, se deben cargar todas las páginas a MP, y resetear el puntero a instrucción.

- Tabla de mapa de archivos: permite convertir direcciones de MV.
  - ↳ solicita bloques de mem. virtual al HDD.

○ Carga dinámica: tratamiento de la excepción del fallo de página

- Fallo de página: se intenta acceder a una página no cargada en MP.

↳ Sol.: guardar estado, cargar página a MP, continuar.

\* Cambio de contexto: cambio de un proceso a otro (con restauración) de registro

- Si la MP está llena; se libera un marco para introducir el otro.

Si el bit de mod. = 1  $\Rightarrow$  Se debe guardar ese marco en HV.

- Pagedores: proc. del SO que realiza la carga dinámica.

↳ maneja páginas entre MV y MP

Tipos:

- De archivos.

- De soportes auxiliares: no tienen imagen en el sistema de archivos.

- De dispositivos.

- Hiperpaginación: caída del rendimiento debido a la multiprogramación.  
↳ Cambio continuo de datos de MV a MP (y viceversa)

Soluciones:

- Disminuir grado de multiprogramación (cada proceso)
- Algoritmo reemplazo local (LRU) (Sob interactiva con sus marcos)
- Suministrar a cada proceso el nº de marcos que necesita:  
- Requiere de espacio

- Soluciones detalladas:

- Frecuencia de fallas de página: un proceso genera muchas fallas de página  
aumenta nº marcos asignados al proceso.

Muchos FP  $\rightarrow$  aumenta nº marcos

Pocos FP  $\rightarrow$  disminuye nº marcos

- Modelos de conjunto de trabajo:

Conjunto de trabajo: grupo de segmentos de MV sobre los que se esté trabajando.

↳ En función de su tamaño, se asignarán más o menos ~~marcos~~  
~~procesos~~

## Algoritmos de Gest. Memoria

Objetivo: minimizar nº fallas de página. (Referencias dentro del proc en RTF)

En base a:

- Políticas de asignación: ¿Qué cantidad de MP se asigna a un proceso?
  - ↳ Asignación fija y variable (fijo = nº marcos fijos // variable = nº marcos variable)
  - ↳ Alcance de reemplazo: global (marcos fijos para todos procesos) o local (marcos fijos para un solo proceso)
  - ↳ Gestión del espacio libre.

- Políticas de ubicación: dónde se ubica un bloque en MP.

• Paginación: da igual pg. Los bloques tienen mismo tamaño

- Segmentación:

- Primer acierto: prima que pillas
- Segundo acierto: siguiente al último.
- Mejor acierto: donde mejor encaje
- Peor acierto: en el hueco más grande

- Políticas de búsqueda: cuándo y qué páginas se cargan en MP.

• Pag. por demanda: sólo se carga en MP cuando se ha referenciado.

↳ En MP sólo hay lo que se necesita, sobre carga informa.

• Pag. anticipada: según un predicción.

• Tiempo de ejecución de procesos reducido.

• Util cuando se accede secuencialmente a disp. de almacenamiento

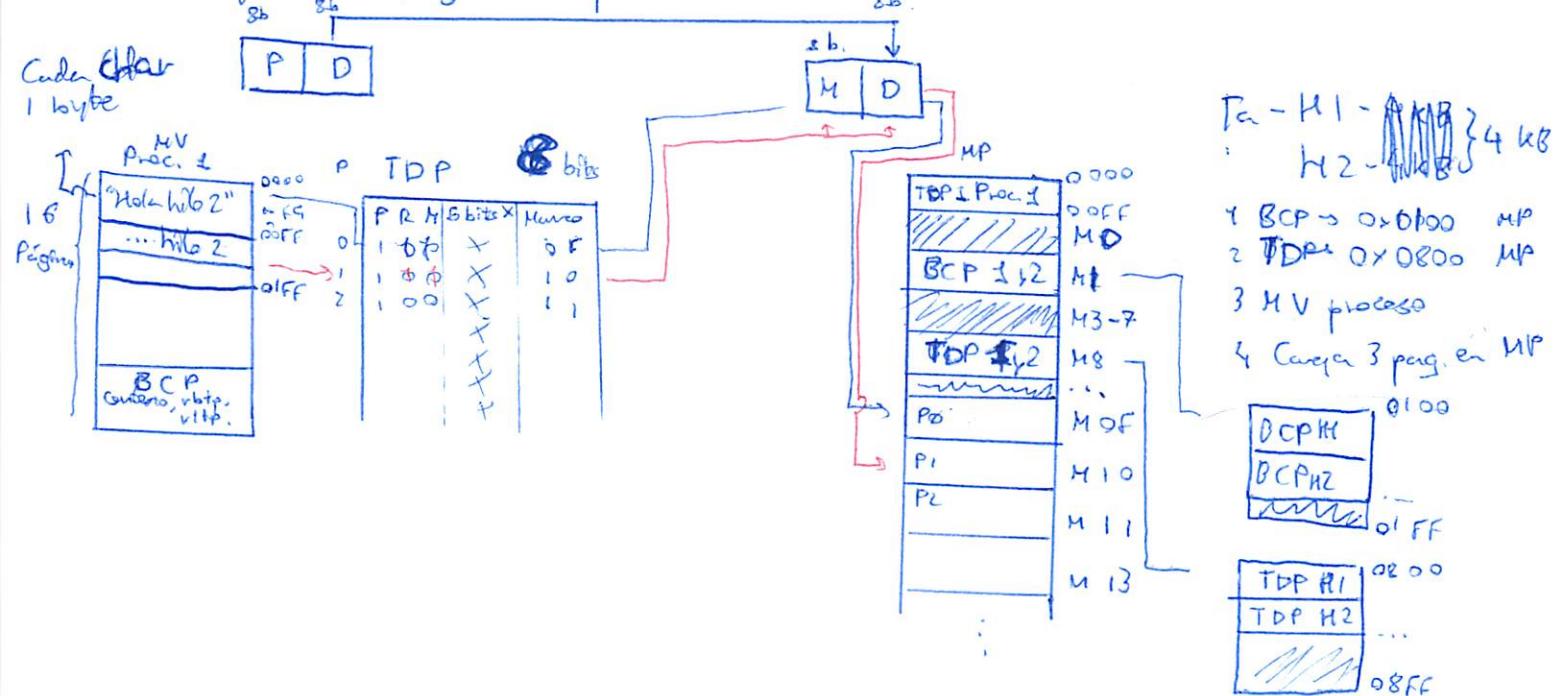
## • Políticas de reemplazo

• Reemplazo local: para un mismo proceso

$$\text{Tam. P} = 2^8 \xrightarrow{\text{desplazamiento}} / 4 \text{ kB} = 2^{12} \text{ Bytes}; 2^{12} / 2^8 = 2^4 \text{ páginas} = 16 \text{ pag.} \rightarrow \text{Tam}$$

1) Ej. Examen 2010. Sist. Op. Val OS. FIFO seg. oportunidad.

Ejemplo de Página en el procesador



• Mach 3.0: FIFO segunda oportunidad

Sigue dentro de los marcos asignados. { 3 colas { Marcos libres: no esté en MP ni en referencia. Cola Páginas libres: no esté en MP ni en referencia. Cola Páginas activas: ∈ conj. trabajo y ref=1 (están en MP). Cola páginas no activas: no ∈ conj. trabajo pero esté en memoria.

Cuando ya no hay libres → FIFO seg. oportunidad.  
Desarrollo: proceso del SO que se ejecuta de forma periódica.

P R

0 0

1 1

1 0

Tarea: entidad que tiene las tareas

↳ Hilos: flujos ejecución

Tam. P =  $2^8$  bytes / 4 kB =  $2^{12}$  Bytes;  $2^{12} / 2^8 = 2^4$  páginas = 16 pag.  $\rightarrow$  Tam

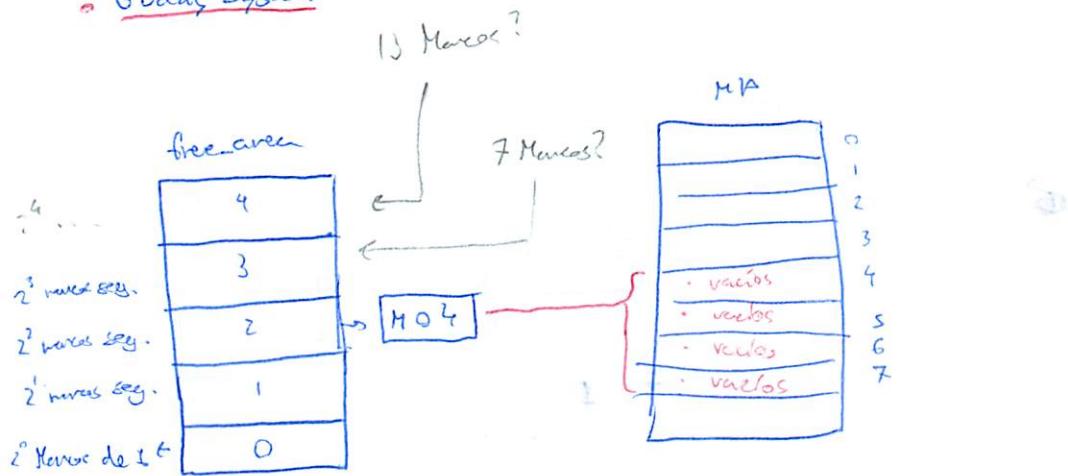
- Taren effericidae \$500 Av.

## ③ Linux

BCP  $\rightarrow$  Task-struct (Linux)

Mmap → Apunta a area-struct  
area-struct → Apunta a sección (código, datos,...) de la m. r. del proceso.

## Buddy System



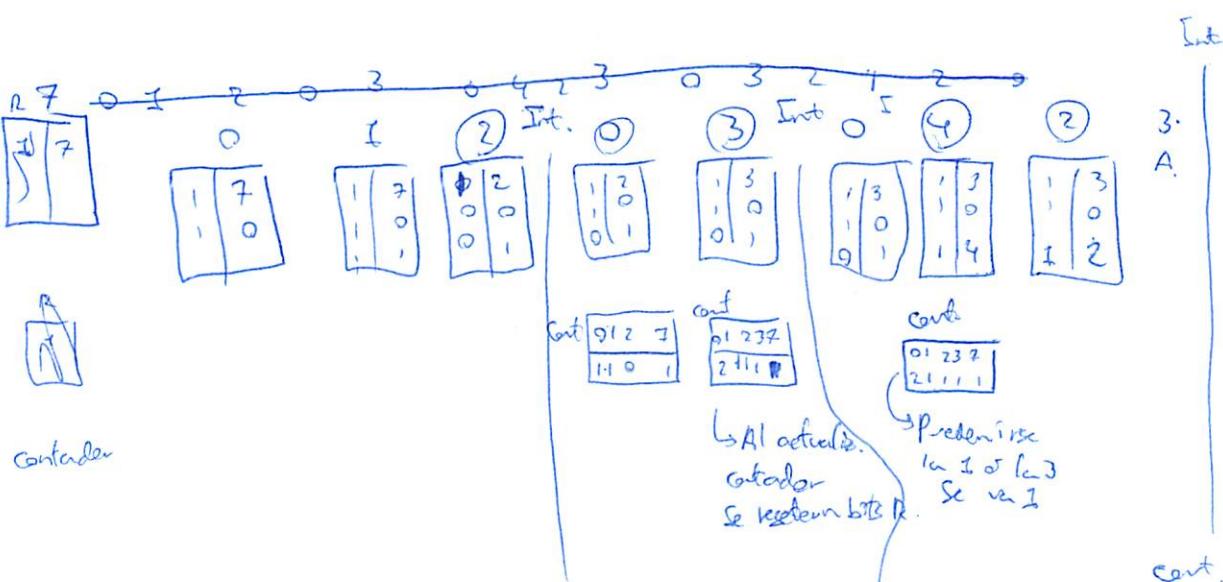
## Algoritmo de reemplazo

Kernel swap daemon

Resumen: **Kernel Swap**  
Funciona: Kernel swap memoria  
La buscan nubes y compara edades de las páginas.  
Si se usa una página de otra nube, se cambia.  
Si no se usa una página de otra nube, se mantiene su vida en 1.

Cuando haya que sustituir, se tomará la que tenga la  
vida más larga.

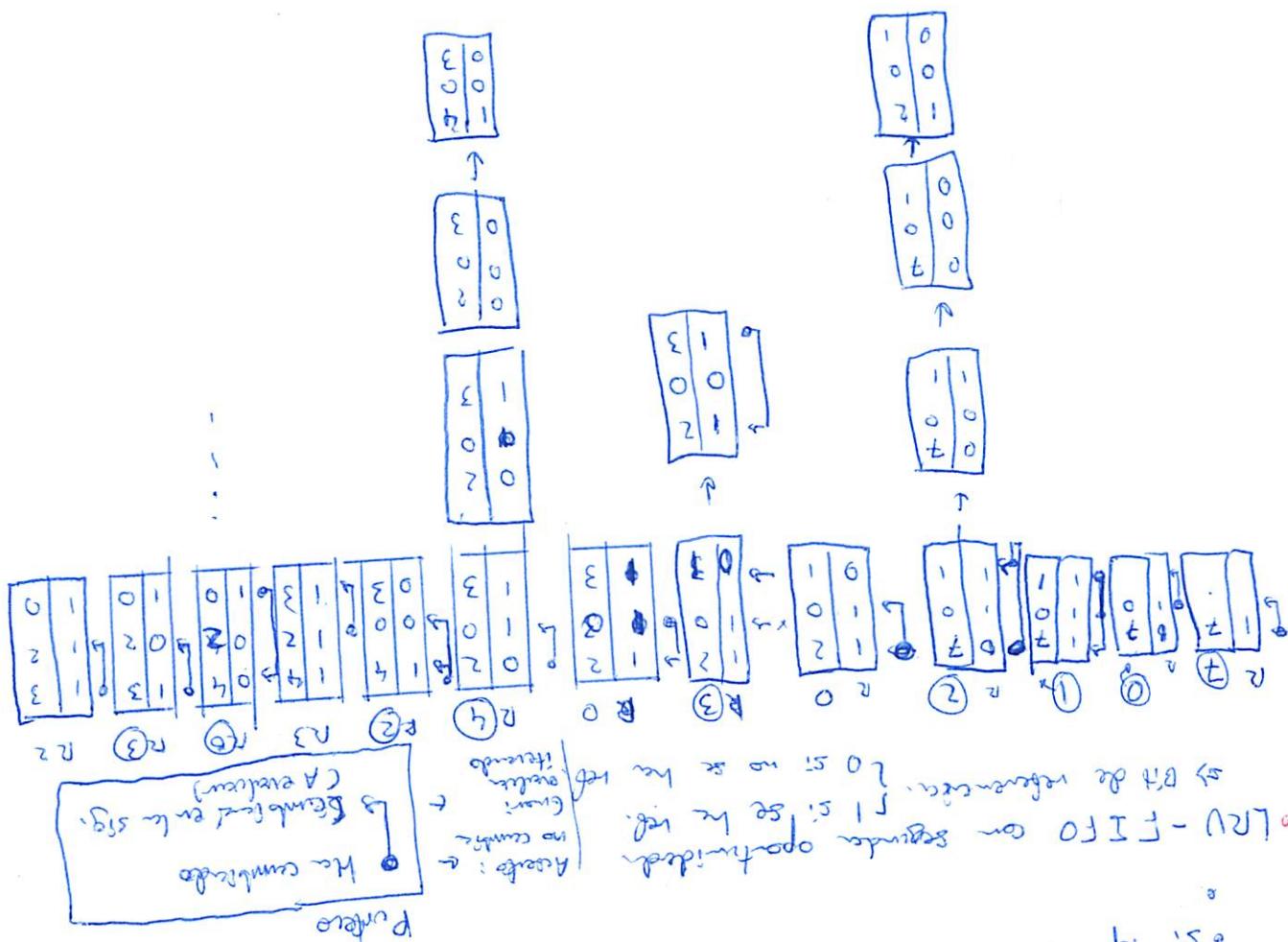
## Algoritmo NFU



Problems: perhaps my readers will have given this sample.

Sol.: en cada siti, desplazar bits de catálogo a la derecha y  
poner por la dg. R.

0	1	2	3	4	5	6	7
3	1	2	2	1	0	0	1



$\Rightarrow$  Cache & CDS are valid, so part of LRU.

$\Rightarrow$  Set size in bit of reference [0 to N] reference

Applies cache LRU. If valid (e.g. reference)

## T2 - SO: Memoria Visual

## Alg. de gestió de memòria

- Políticas de reemplazo: decide qué páginas se sustituyen en MP.
    - $\Rightarrow$  Algoritmo óptimo: se reemplaza la página que va a tardar más tiempo en usarse.
    - No implementable. Usando como p.t.o. referencia óptima como el resto de alg.
  - Ubicación dinámica: el marco queda libre y se mete otro.
    - Asignación estatística: tamaño fijo cantidad de marcos cte.
    - Asignación dinámica: tamaño y cantidad de marcos variable.

Report

- $\Rightarrow$  FIFO: se reemplaza la página que lleva más tiempo en MP.

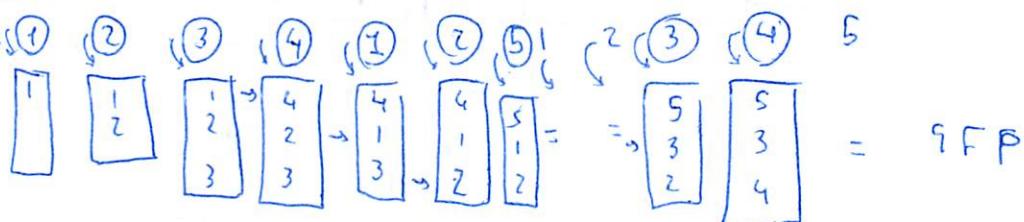
- Ventajas: sencillo de implementar

### Inconvenientes:

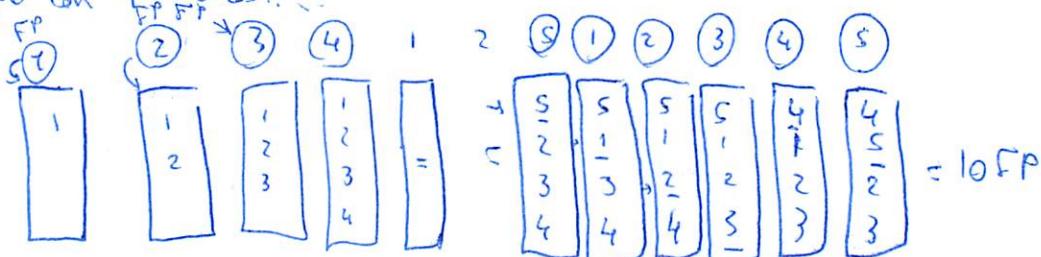
- Enige trends waarden

- Anomalia de Belady: aumento de FPa al aumentar n° mareas.

Exemples: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮

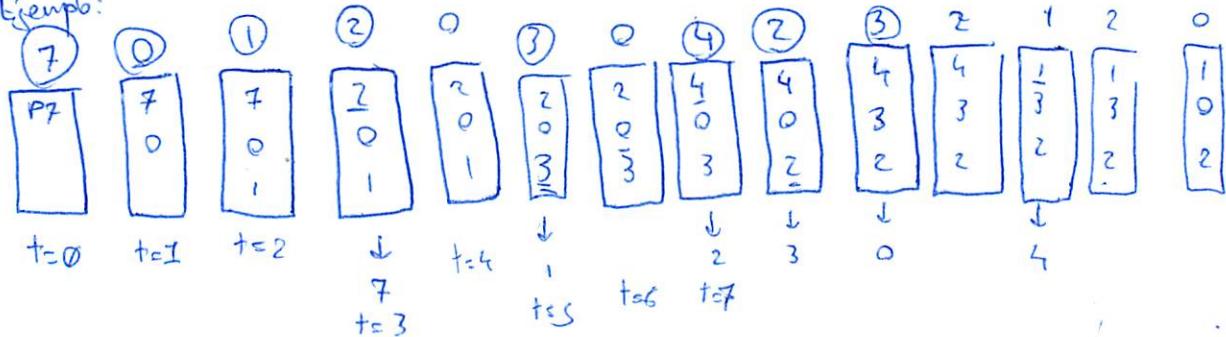


Ejemplo con 4 puentes.



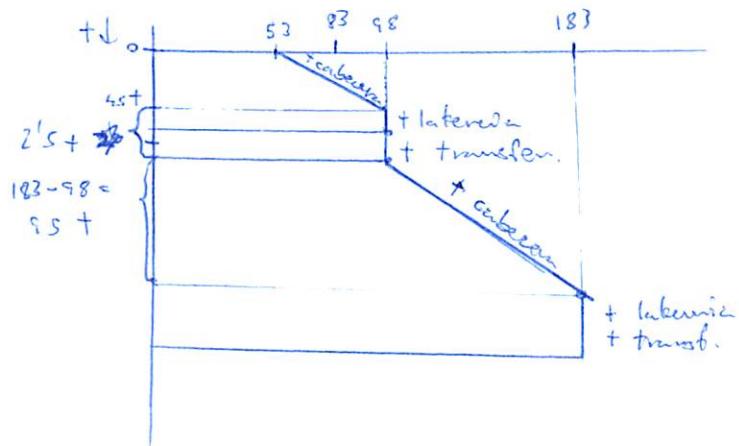
- $\Rightarrow$  Algoritmo LRU: se añade a la tabla de páginas una cédula de bits "timestamp". Utiliza el pasado reciente para predecir el futuro.  
Sustituye la página menos usada en el pasado.
    - Ventajas: Aproximación al óptimo.
    - Inconvenientes:
      - Dificultad de implementación
      - Alta sobre carga
      - Selección: algoritmos de aproximación al LRU.

Ejemplo:



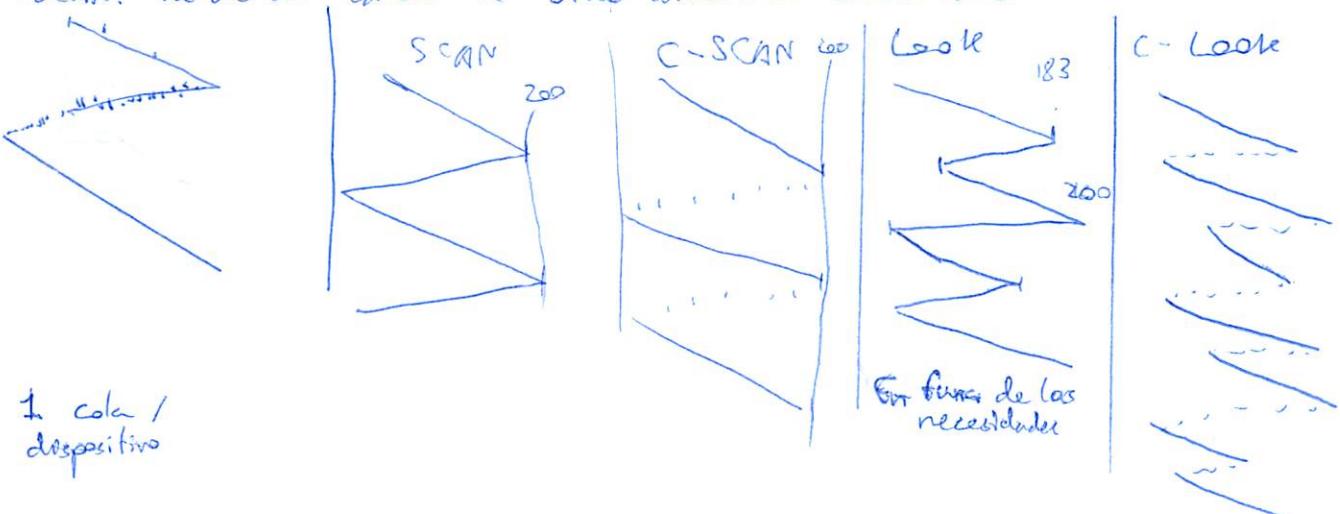
## ◦ Planiificación E/S de Disco

◦ FCFSS: primera en llegar, primera en ser servida.



◦ SSTF: nos movemos a las pistas más cercanas.  
Problema: cuando hay solicitudes muy lejanas se produce inmovilización.

◦ SCAN: va de un extremo a otro atendiendo solicitudes



## Cap 3 SSOO. - Gestión de I/O

Dispositivos I/O: teclado, ratón, monitor, tarjeta de red, etc.

Cada uno funciona de forma diferente y se accede a ellos de forma diferente.

- Interfaz de usuario: interruptores
- Disco duro: Direct Memory Access

### Gestión de I/O

Manejador: gestiona el dispositivo a nivel de Software (Drivers)

El usuario utiliza una interfaz de uniforme acceso: todos los dispositivos se manejan de la misma forma. (I/O independiente del dispositivo).

Investigan de la misma forma. (I/O independiente del dispositivo).

Los SO modernos son plug and play: no hace falta reiniciar el ordenador para que detecte el dispositivo.

Todos los dispositivos en UNIX son archivos.

### Elementos I/O

Cada elemento se conecta directamente (North Bridge, South Bridge)

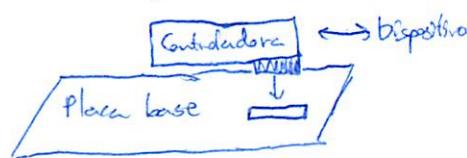
Dividido en 2 partes:

- Controladora o adaptador de dispositivo:

- Circuito o dispositivo integrado conectado a ranura de expansión (zócalo) del ordenador. Con reg. estado, control y datos.

- El dispositivo en sí:

- Conectado mediante un zócalo o cable.

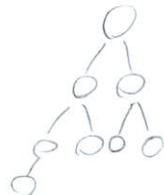
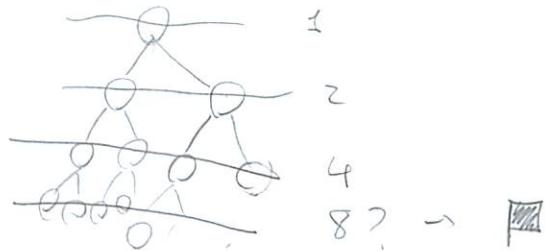


### Manejadores de interrupciones

El SO realiza varias tareas:

- Guarda registros y pwr y prepara un contexto y una pila para la INT.
- Envía señales a controladora de interrupciones.
- Ejecuta la ISR.

Apellidos:	Pág.:
Nombre:	Fecha:
Titulación:	
Asignatura:	Curso / grupo:



Contador  
[ ]

VAR

Nodos : Cola  
Nodos : Cola  
RAICES : lista

Contador = 0  
encolar(nodos, a)  
raices : primero(cola) ^ . raiz  
mientras (! vacia(cola)) :  
    raices & [ ]  
    Contador x = 2  
    mientras (cola(cola)) :  
        abb & desencolar(nodos)  
        | si (abb^.izq != null) encolar (nodos2, abb^.izq)  
        | si (abb^.der != null) encolar (nodos2, abb^.der)

mientras

insertar (raiz(cola(cola2))) :

    extruido & desencolar (nodos2)

    raices: extruido^. raiz

    encolar (nodos2, extruido)

f mientras

si longitud(raices) != contador n

    \ vuelo (nodos) / der +

    sinbot (sc)

der +