Sistemas Operativos

Universidad Complutense de Madrid

2020-2021

Sistemas de ficheros

Soluciones de problemas

Juan Carlos Sáez

Problema 2

- ¿Número de accesos a disco (*bloques*) requeridos para acceder 20 bloques lógicos consecutivos de un fichero?
  - Número de bloque $\in [k \ldots k+19]$
- 3 sistemas de ficheros distintos
  1. Asignación contigua (CDROM)
  2. Asignación enlazada (FAT)
  3. Asignación indexada (UNIX)
- Determinar el número de accesos requeridos en el caso mejor y el caso peor

Problema 2: Suposiciones

Suposiciones

1. El fichero está ubicado en el directorio de trabajo (actual) del proceso
2. El descriptor físico del directorio de trabajo está cargado en memoria
   - Optimización típica en el SO
3. No hay más datos del sistema de ficheros cargados en memoria

Problema 2: Notación auxiliar

Notación

- $N \equiv$ Número de entradas del directorio actual
- $T_{entrada} \equiv$ Tamaño (en bytes) de una entrada de directorio
- $T_{bloque} \equiv$ Tamaño (en bytes) de un bloque del sistema de ficheros
  - Asumiremos que $T_{entrada} \ll T_{bloque}$
  - Por tanto $\left\lceil \frac{T_{entrada}}{T_{bloque}} \right\rceil = 1$

Problema 2(a): Asignación contigua

Características del SF (p.ej., CDROM)

- Descriptor físico: entrada de directorio
- El SO localiza los bloques lógicos de un fichero en disco empleando el par (*DirPrimerBloqueLog*, *NumBloques*)
  - Este par se almacena en el descriptor físico del fichero

2 pasos para acceder a los bloques

1. Encontrar el fichero en el directorio
   - Accedemos al descriptor físico
2. Acceder a los bloques de datos (consecutivos y contiguos)

Problema 2(a): Asignación contigua

Caso peor

- El fichero está ubicado al final del directorio

$$\#\text{ de accesos a disco} = \left\lceil \frac{N * T_{entrada}}{T_{bloque}} \right\rceil + 20$$

Caso mejor

- El fichero está ubicado al principio del directorio (primer bloque)

$$\#\text{ de accesos a disco} = 1 + 20$$

Problema 2(b): Asignación enlazada

Características del SF FAT

- Descriptor físico: entrada de directorio
- El SO mantiene una lista enlazada de bloques para llevar la cuenta de la ubicación física de los bloques de un fichero en disco
  - La dirección del primer bloque lógico se encuentra en el descriptor físico del fichero
  - El resto de direcciones se almacenan en la FAT (*File Allocation Table*)

3 pasos para acceder a los bloques

1. Encontrar el fichero en el directorio
   - Accedemos al descriptor físico
2. Recorrer lista enlazada de bloques en la FAT para obtener dirección física
3. Acceder a los bloques de datos

Problema 2(b): Asignación enlazada

Caso peor (3 condiciones)

- El fichero está ubicado al final del directorio
- Los bloques lógicos referenciados están al final del fichero
  - Se ha de recorrer la lista enlazada completa
- Los 20 bloques lógicos consecutivos están desperdigados por el disco
  - Esto requeriría acceder a la FAT completa para realizar la traducción

$$\#\text{ de accesos a disco} = \left\lceil \frac{N * T_{entrada}}{T_{bloque}} \right\rceil + \text{Tam}_{\text{FAT}} + 20$$

Problema 2(b): Asignación enlazada

Caso mejor (3 condiciones)

- El fichero está ubicado al principio del directorio (primer bloque)
- Los bloques lógicos referenciados son los 20 primeros del fichero
- Los 20 bloques lógicos consecutivos están almacenados en 20 bloques físicos contiguos
  - En el mejor escenario esto requeriría acceder a un único bloque de disco (de la FAT) para obtener la dirección física de los 20 bloques

$$\#\text{ de accesos a disco} = 1 + 1 + 20$$

Problema 2(c): Asignación indexada

Características del SF UNIX

- Descriptor físico: Nodo-i
- Los bloques de datos de un fichero se referencian mediante punteros
  - Asumiremos una organización tradicional
    - 10 punteros directos
    - 1 puntero indirecto simple
    - 1 puntero indirecto doble
    - 1 puntero indirecto triple
- Los directorios se modelan como tablas de pares (nombre, núm. nodo-i)
  - Se requiere un acceso extra a disco para recuperar el nodo-i (acceso a tabla de nodos-i)

Problema 2(c): Asignación indexada

4 pasos para acceder a los bloques

1. Encontrar el fichero en el directorio
   - Recuperamos únicamente el número de nodo-i
2. Se requiere un acceso extra a disco para recuperar el nodo-i (acceso a tabla de nodos-i)
3. Para cada bloque lógico, determinar su ubicación física accediendo a los punteros de forma directa o indirecta a partir de la información en el nodo-i
4. Acceder a los bloques de datos

Problema 2(c): Asignación indexada

Caso peor (2 condiciones)

- El fichero está ubicado al final del directorio
- Los bloques lógicos a los que se accede están en el nivel indirecto triple
  - Conlleva el acceso a hasta 4 bloques de índices

$$\#\text{ de accesos a disco} = \left\lceil \frac{N * T_{entrada}}{T_{bloque}} \right\rceil + 1 + 4 + 20$$

Problema 2(c): Asignación indexada

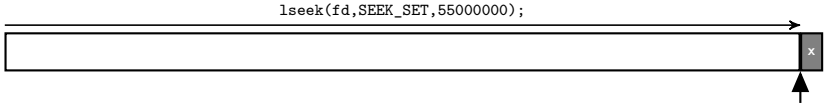
Caso mejor (2 condiciones)

- El fichero está ubicado al principio del directorio (primer bloque)
- Los bloques referenciados son los 20 primeros bloques lógicos del fichero
  - Requiere acceder a un bloque de índices para recuperar la dirección física de los bloques lógicos del 10 al 19
  - Las direcciones físicas de los 10 primeros bloques lógicos (del 0 al 9) están almacenadas en el nodo-i

$$\#\text{ de accesos a disco} = 1 + 1 + 1 + 20$$

Problema 6

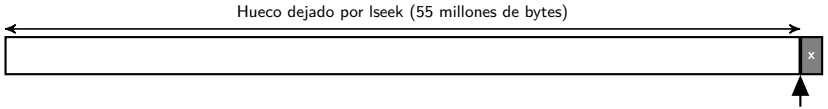
1. Un programa abre un fichero con `open()`
   - El fichero está inicialmente vacío
2. Se posiciona en el byte 55 millones con `lseek()`
3. Escribe un byte
   - `write(fd,"x",1)`



Problema 6

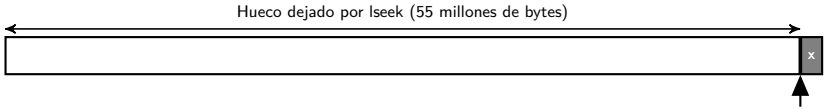
- Proporcionar una respuesta a las siguientes preguntas
  1. *¿Cuántos bloques de disco ocupa ahora el fichero (incluyendo bloques indirectos)?*
  2. *¿Qué sucesión de índices lógicos nos lleva al byte posicionado por lseek?*
- El fichero se ha creado en un SF UNIX...
  - T_{bloque}=2KB
  - T_{puntero} = 32 bits = 4 bytes
  - Nodos-i:
    - 10 punteros directos
    - 1 puntero indirecto simple
    - 1 puntero indirecto doble
    - 1 puntero indirecto triple

Hueco creado por `lseek()`



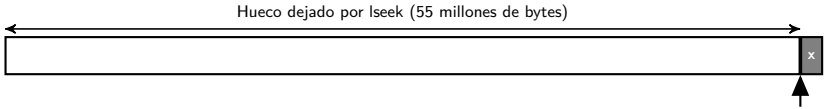
- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero

Hueco creado por `lseek()`



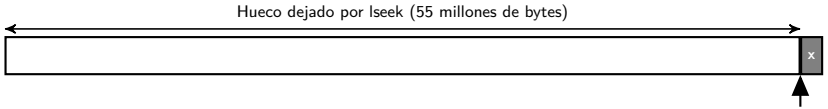
- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero
- Surgen **dos preguntas** sobre este hueco

Hueco creado por `lseek()`



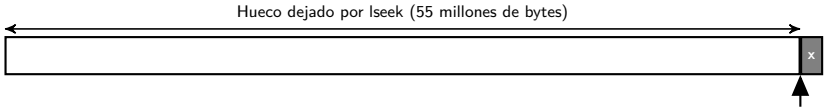
- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero
- Surgen **dos preguntas** sobre este hueco
  1. *¿Qué ocurre si leemos un byte del hueco? ¿Qué devuelve `read()`?*

Hueco creado por `lseek()`



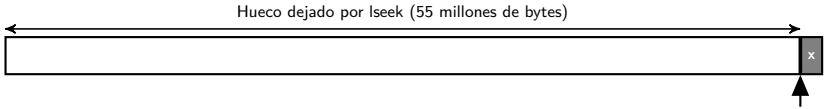
- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero
- Surgen **dos preguntas** sobre este hueco
  1. *¿Qué ocurre si leemos un byte del hueco? ¿Qué devuelve `read()`?*
  2. *¿Ocupan espacio en disco los bytes que hay en el hueco?*

Hueco creado por `lseek()`



- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero
- Surgen **dos preguntas** sobre este hueco
  1. *¿Qué ocurre si leemos un byte del hueco? ¿Qué devuelve `read()`?*
  2. *¿Ocupan espacio en disco los bytes que hay en el hueco?*
- Para obtener la respuesta a la primera pregunta: `$ man 2 lseek`

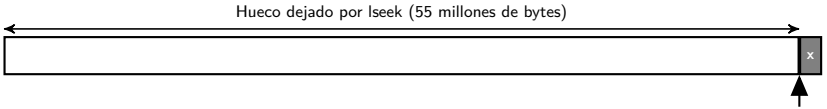
Hueco creado por `lseek()`



- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero
- Surgen **dos preguntas** sobre este hueco
  1. *¿Qué ocurre si leemos un byte del hueco? ¿Qué devuelve `read()`?*
  2. *¿Ocupan espacio en disco los bytes que hay en el hueco?*
- Para obtener la respuesta a la primera pregunta: `$ man 2 lseek`

  El estándar POSIX establece que el valor que obtenemos al leer cualquier byte del un "hueco" es 0

Hueco creado por `lseek()`



- Al mover el puntero de posición más allá del fin de fichero, y a continuación escribir un byte, el programa crea un *hueco* enorme en el fichero
- Surgen **dos preguntas** sobre este hueco
  1. *¿Qué ocurre si leemos un byte del hueco? ¿Qué devuelve `read()`?*
  2. *¿Ocupan espacio en disco los bytes que hay en el hueco?*
- Para obtener la respuesta a la primera pregunta: `$ man 2 lseek`

  El estándar POSIX establece que el valor que obtenemos al leer cualquier byte del un "hueco" es 0

  $\Rightarrow$ Sin embargo, esto no significa que haya que almacenar 0s en disco para implementar este comportamiento

Alternativas de implementación

El SO tiene dos opciones para implementar el hueco

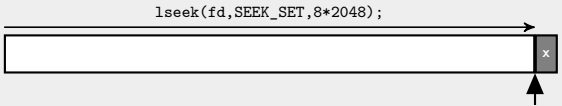
1. **Usar punteros a NULL en el nodo-i** para representar bloques lógicos no utilizados (aquellos que se encuentran en el hueco)
   - No se desperdicia espacio en disco
     - Si un programa de usuario lee cualquier byte del hueco con `read(fd,buffer,nbytes)` → `buffer` se llena con ceros
2. **Reservar bloques de disco para los bytes del hueco** y **rellenar estos bloques con 0s**
   - Ineficiente en términos de espacio

- La opción 1 es la más adecuada para sistemas de ficheros basados en nodos-i
  - Asumiremos esta alternativa de implementación
- Para SSFF con organización enlazada (FAT) solo es posible la opción 2

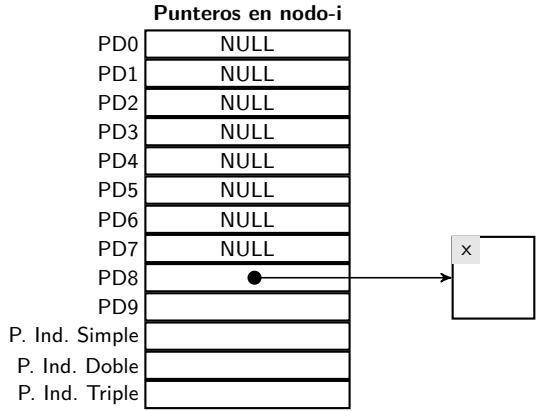
Alternativas de implementación: ejemplo

Ejemplo más simple

1. Un programa abre un fichero con `open()`
   - El fichero está inicialmente vacío
2. Se posiciona en el byte 16K con `lseek()`
   - $T_{\text{bloque}} = 2KB \Longrightarrow$ byte 16K es Bloque lógico $\#8$
3. Escribe un byte
   - `write(fd,"x",1)`

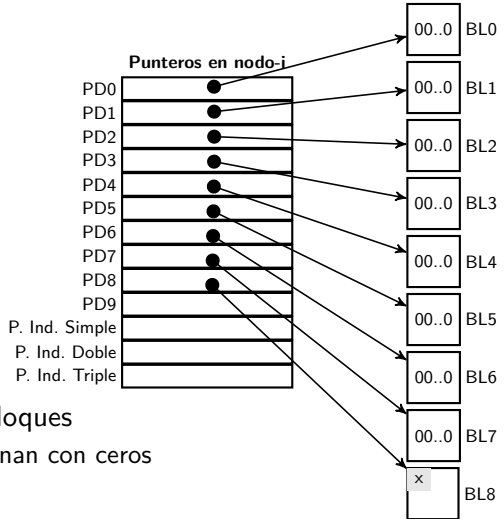


Alternativas de implementación: opción 1



El fichero ocupa solo un bloque en disco

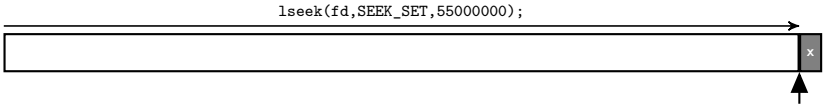
Alternativas de implementación: opción 2



- El fichero ocupa 9 bloques
  - 8 bloques se rellenan con ceros

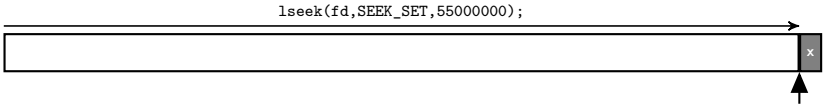
Problema 6.a

- *¿Cuántos bloques de disco ocupa ahora el fichero (incluyendo bloques indirectos)?*
  - Asumiremos que se usa la Opción 1 de implementación para el hueco



Problema 6.a

- *¿Cuántos bloques de disco ocupa ahora el fichero (incluyendo bloques indirectos)?*
  - Asumiremos que se usa la Opción 1 de implementación para el hueco



- Se ha de determinar primero qué bloque lógico almacena el byte 55M

Dir. lógica $\rightarrow$ Dir. bloque lógico = (Núm. bloque lógico, Offset)

$$\text{Num. bloque lógico} = \left\lfloor \frac{\text{DirLógica(bytes)}}{T_{\text{bloque}}(bytes)} \right\rfloor = \left\lfloor \frac{55 \cdot 10^6}{2048} \right\rfloor = 26855$$

$$\text{Offset} = \text{DirLógica(bytes)} \ \textbf{mod} \ T_{\text{bloque}}(bytes) = 55 \cdot 10^6 \ \textbf{mod} \ 2048 = 960$$

Problema 6.a

- Ahora debemos averiguar el nivel de punteros en el nodo-i usado para apuntar al bloque lógico 26855 (directo, indirecto simple, ….)
  - ¿Cuántos punteros caben en un bloque de índices?
    - $N_{\text{punterosPorBloqueInd}} = \left\lfloor \frac{T_{bloque}(bytes)}{T_{puntero}(bytes)} \right\rfloor = \left\lfloor \frac{2048}{4} \right\rfloor = \left\lfloor \frac{2^{11}}{2^2} \right\rfloor = 512$ punteros

Problema 6.a

- Ahora debemos averiguar el nivel de punteros en el nodo-i usado para apuntar al bloque lógico 26855 (directo, indirecto simple, ….)
  - ¿Cuántos punteros caben en un bloque de índices?
    - $N_{\text{punterosPorBloqueInd}} = \left\lfloor \frac{T_{bloque}(bytes)}{T_{puntero}(bytes)} \right\rfloor = \left\lfloor \frac{2048}{4} \right\rfloor = \left\lfloor \frac{2^{11}}{2^2} \right\rfloor = 512$ punteros
- *¿Nivel directo?* $\rightarrow$ 10 punteros directos. Pero $10 \ll 26855$

Problema 6.a

- Ahora debemos averiguar el nivel de punteros en el nodo-i usado para apuntar al bloque lógico 26855 (directo, indirecto simple, ….)
  - ¿Cuántos punteros caben en un bloque de índices?
    - $N_{\text{punterosPorBloqueInd}} = \left\lfloor \frac{T_{bloque}(bytes)}{T_{puntero}(bytes)} \right\rfloor = \left\lfloor \frac{2048}{4} \right\rfloor = \left\lfloor \frac{2^{11}}{2^2} \right\rfloor = 512$ punteros
- *¿Nivel directo?* $\rightarrow$ 10 punteros directos. Pero $10 \ll 26855$
- *¿Nivel indirecto simple?* $\rightarrow 512 + 10 \ll 26855$

Problema 6.a

- Ahora debemos averiguar el nivel de punteros en el nodo-i usado para apuntar al bloque lógico 26855 (directo, indirecto simple, ….)
  - ¿Cuántos punteros caben en un bloque de índices?
    - $N_{\text{punterosPorBloqueInd}} = \left\lfloor \frac{T_{bloque}(bytes)}{T_{puntero}(bytes)} \right\rfloor = \left\lfloor \frac{2048}{4} \right\rfloor = \left\lfloor \frac{2^{11}}{2^2} \right\rfloor = 512$ punteros
- *¿Nivel directo?* $\rightarrow$ 10 punteros directos. Pero $10 \ll 26855$
- *¿Nivel indirecto simple?* $\rightarrow 512 + 10 \ll 26855$
- *¿Nivel indirecto doble?*
  - Dos niveles de bloques de índices: 512^2 punteros $= 262144$ punteros

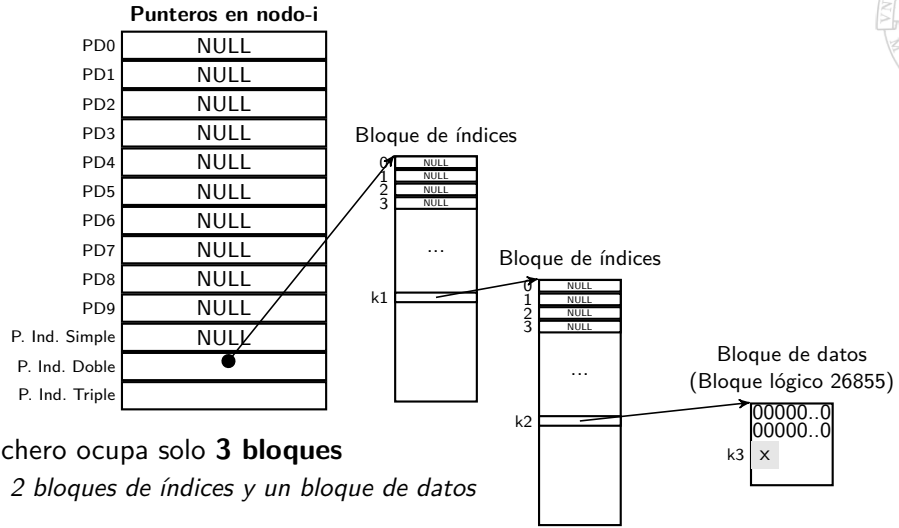
Problema 6.a

- Ahora debemos averiguar el nivel de punteros en el nodo-i usado para apuntar al bloque lógico 26855 (directo, indirecto simple, ….)
  - ¿Cuántos punteros caben en un bloque de índices?
    - $N_{\text{punterosPorBloqueInd}} = \left\lfloor \frac{T_{bloque}(bytes)}{T_{puntero}(bytes)} \right\rfloor = \left\lfloor \frac{2048}{4} \right\rfloor = \left\lfloor \frac{2^{11}}{2^2} \right\rfloor = 512$ punteros
- *¿Nivel directo?* $\rightarrow$ 10 punteros directos. Pero $10 \ll 26855$
- *¿Nivel indirecto simple?* $\rightarrow 512 + 10 \ll 26855$
- *¿Nivel indirecto doble?*
  - Dos niveles de bloques de índices: 512^2 punteros $= 262144$ punteros
  - $262144 + 512 + 10 \gg 26855$

Problema 6.a

- Ahora debemos averiguar el nivel de punteros en el nodo-i usado para apuntar al bloque lógico 26855 (directo, indirecto simple, ….)
  - ¿Cuántos punteros caben en un bloque de índices?
    - $N_{\text{punterosPorBloqueInd}} = \left\lfloor \frac{T_{bloque}(bytes)}{T_{puntero}(bytes)} \right\rfloor = \left\lfloor \frac{2048}{4} \right\rfloor = \left\lfloor \frac{2^{11}}{2^2} \right\rfloor = 512$ punteros
- *¿Nivel directo?* $\rightarrow$ 10 punteros directos. Pero $10 \ll 26855$
- *¿Nivel indirecto simple?* $\rightarrow 512 + 10 \ll 26855$
- *¿Nivel indirecto doble?*
  - Dos niveles de bloques de índices: 512^2 punteros $= 262144$ punteros
  - $262144 + 512 + 10 \gg 26855$
  - **El byte 55 millones se encuentra en el nivel indirecto doble**

Problema 6.a



- El fichero ocupa solo **3 bloques**
  - *2 bloques de índices y un bloque de datos*

Problema 6.b

- *¿Qué sucesión de índices lógicos nos lleva al byte posicionado por lseek?*
  - Los índices lógicos son $(k1, k2, k3)$
  - k3 es el desplazamiento (offset) dentro del bloque $\rightarrow k3 = 960$

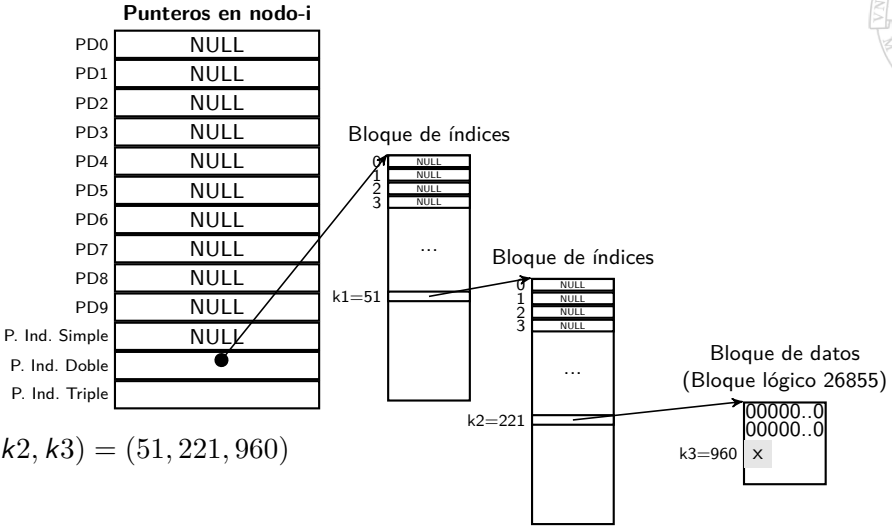
Problema 6.b

- *¿Qué sucesión de índices lógicos nos lleva al byte posicionado por lseek?*
  - Los índices lógicos son $(k1, k2, k3)$
  - k3 es el desplazamiento (offset) dentro del bloque $\rightarrow k3 = 960$
- Primero debemos determinar el número relativo asociado al bloque lógico 26855 dentro del nivel indirecto doble
  - Restar número de punteros presentes en niveles anteriores
  - *NumRelativoBloque*$= 26855 - 512 - 10 =$ 26333

Problema 6.b

- *¿Qué sucesión de índices lógicos nos lleva al byte posicionado por lseek?*
  - Los índices lógicos son $(k1, k2, k3)$
  - k3 es el desplazamiento (offset) dentro del bloque $\rightarrow k3 = 960$
- Primero debemos determinar el número relativo asociado al bloque lógico 26855 dentro del nivel indirecto doble
  - Restar número de punteros presentes en niveles anteriores
  - *NumRelativoBloque*$= 26855 - 512 - 10 = 26333$
- Cada puntero en el primer bloque de índices usado apunta a un grupo de 512 punteros (en otro bloque de índices). Por tanto tenemos que:
  - $k1 = \left\lfloor \frac{\textit{NumRelativoBloque}}{N_{\text{punterosPorBloqueInd}}} \right\rfloor = \left\lfloor \frac{26333}{512} \right\rfloor = 51$
  - $k2 = \textit{NumRelativoBloque}$ **mod** $N_{\text{punterosPorBloqueInd}} = 221$

Problema 6.b



- $(k1, k2, k3) = (51, 221, 960)$

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7

bloque 12

bloque 15: Datos sin formato

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7

bloque 12

bloque 15: Datos sin formato

En UNIX el nodo-i 2 corresponde al directorio raíz ("/"). Además, por definición, el padre de "/" es '/'. Podemos rellenar la entrada ".." en el bloque 3, que es el del directorio raíz en este caso.

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7

bloque 12

bloque 15: Datos sin formato

El directorio almacenado en el bloque 0 tiene nodo-i asociado número 5 (entrada "." del directorio). Podemos completar el campo Tamaño del nodo-i 5 (1 bloque).

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	5
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7

bloque 12

bloque 15: Datos sin formato

El bloque 5 almacena un directorio cuyo nodo-i es el 9 (entrada "." del directorio). Podemos completar el campo Índice Directo del nodo-i 9 (valor=5).

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	5
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7:

bloque 12: Datos sin formato

bloque 15: Datos sin formato

El nodo-i 4 es un fichero regular cuyos datos están en el bloque #12. Podemos completar el contenido del bloque con "Datos sin formato".

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	2
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	5
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7:

bloque 12: Datos sin formato

bloque 15: Datos sin formato

El nodo-i 4 se referencia en los directorios almacenados en los bloques 0 y 3. Por tanto, el fichero tiene 2 alias. Podemos rellenar el campo contador de enlaces en el nodo-i 4.

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	2
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	5
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6: Datos sin formato

bloque 7: 15

bloque 12: Datos sin formato

bloque 15: Datos sin formato

El bloque #7 es un bloque de índices (según el contenido del nodo-i 3) pero desconocemos su contenido. Analizando qué bloques marcados como ocupados están sin referenciar por los nodos-i vemos que sólo queda uno: el 15. Por lo tanto el bloque de índices 7 almacena un puntero al bloque de datos 15.

Problema 8.a

Mapa de bits: 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0

nodo-i 2

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	3
Indirecto	Null

nodo-i 3

Tamaño	2
#Enlaces	1
Tipo F/D	F
Directo	6
Indirecto	7

nodo-i 4

Tamaño	1
#Enlaces	2
Tipo F/D	F
Directo	12
Indirecto	Null

nodo-i 5

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	0
Indirecto	Null

nodo-i 9

Tamaño	1
#Enlaces	NA
Tipo F/D	D
Directo	5
Indirecto	Null

bloque 0

.	5
..	2
C	9
D	4

bloque 3

.	2
..	2
A	3
B	5
E	4

bloque 5

.	9
..	5

bloque 6

Datos sin formato

bloque 7

15

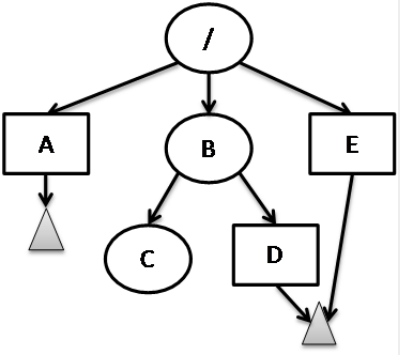
bloque 12

Datos sin formato

bloque 15

Datos sin formato

Problema 8.b



Problema 9.a

- Bloques de 1024B, direcciones de 16 bits (2B) $\Rightarrow N_{indicesBloque} = \lfloor \frac{T_{bloque}}{T_{indice}(bytes)} \rfloor = 512$ índices por bloque
- Offset y tamaño se representan con 64 bits

Tamaño máximo del fichero

- Por organización :

$T_{max\_org} = T_{bloque} * (8 + 512 + (512)^2) \approx 256{,}5$ MB

- Por tamaño de índice:

$T_{max\_tamInd} = 2^{T_{indice}(bits)} \cdot T_{bloque} = 2^{16} * 2^{10} = 64$ MB

- Por tamaño de offset y campo tamaño de fichero:

$T_{max\_offset} = 2^{T_{offset}(bits)}\, bytes = 2^{64} bytes = 16$ exabytes

- Tamaño máximo

$min(T_{max\_org}, T_{max\_tamInd}, T_{max\_offset}) = 64$ MB

Problema 9.b

- Suponer ahora direcciones de 32 bits (4B) $\Rightarrow$

$N_{indicesBloque} = \lfloor \frac{T_{bloque}}{T_{indice}(bytes)} \rfloor = 256$ índices por bloque

Tamaño máximo del fichero

- Por organización :

$T_{max\_org} = T_{bloque} * (8 + 256 + (256)^2) \approx$ 64,26 MB

- Por tamaño de índice:

$T_{max\_tamInd} = 2^{T_{indice}(bits)} \cdot T_{bloque} = 2^{34} * 2^{10} = 2^{42} =$ 4 terabytes

- Por tamaño de offset y campo tamaño de fichero:

$T_{max\_offset} = 2^{T_{offset}(bits)}\, bytes = 2^{64}\, bytes = 16$ exabytes

- Tamaño máximo

$min(T_{max\_org}, T_{max\_tamInd}, T_{max\_offset}) =$ 64,26 MB

Problema 9.c (Ruta:/John/Notes/mem)

Pasos

- Traer a memoria nodo-i 1 (dir. raíz en este caso!!)

Tam (bloques):	1
F/D:	D
Enlaces:	N/A
Ind. directo:	4

- Traer a memoria el bloque 4 (directorio)

Nombre	Nodo-i
.	1
..	1
Mark	2
John	3
phones	4

- Traer a memoria nodo-i 3

Tam (bloques):	1
F/D:	D
Enlaces:	N/A
Ind. directo:	6

Problema 9.c (II)

Pasos (cont.)

- Traer a memoria el bloque 6 (directorio)

Nombre	Nodo-i
.	3
..	1
Notes	7
Temp	8

- Traer a memoria nodo-i 7

Tam (bloques):	1
F/D:	D
Enlaces:	N/A
Ind. directo:	9

Problema 9.c (III)

Pasos (cont.)

- Traer a memoria el bloque 9 (directorio)

Nombre	Nodo-i
.	7
..	3
mem	10

- Traer a memoria nodo-i 10 (FIN)

Tam (bloques):	1
F/D:	F
Enlaces:	2
Ind. directo:	12