

IIC2333 - T2

Memoria

Es un archivo como  
cualquier otro.

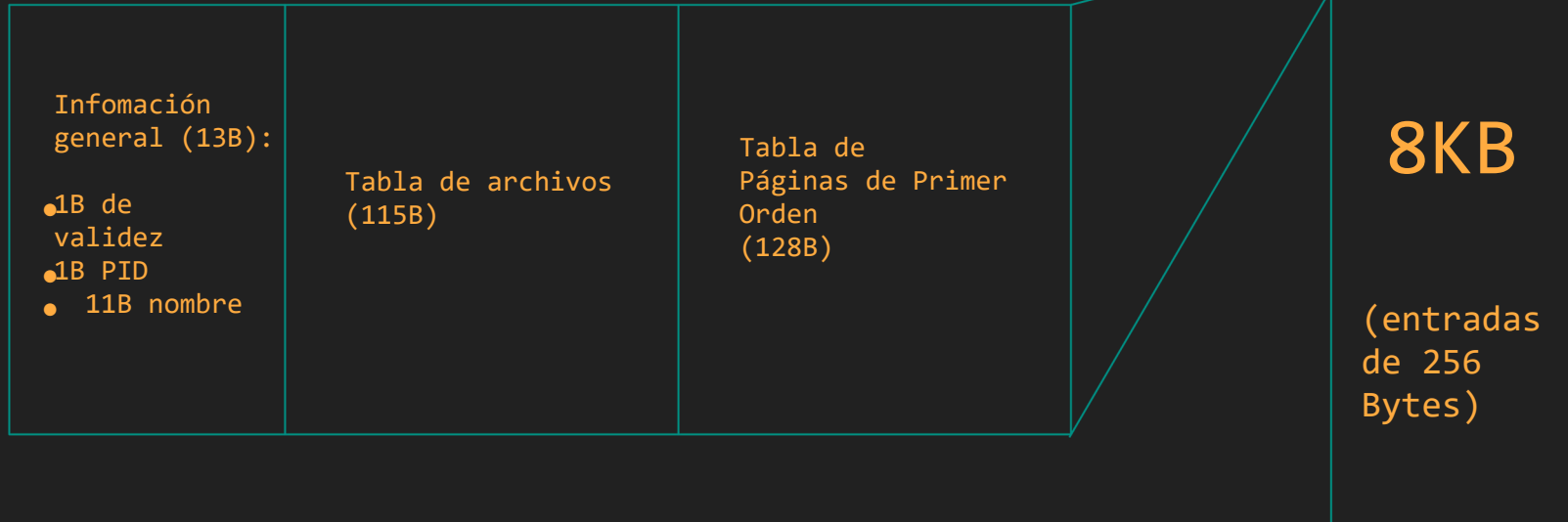
# Memoria

Tabla de PCBs  8KB	Bitmap de TPSO 128 B	Espacio de TPSO 128 KB	Frame bitmap 8 KB				2 GB ... (Frames de 32 KB )
-----------------------------	-------------------------------	------------------------------	-------------------------	--	--	--	--------------------------------

# Tabla de PCBs

# Tabla de PCBs

- 32 entradas ( $32 \times 256\text{B} = 8192\text{B} = 8\text{KB}$ ).
- Cada entrada tiene información sobre un proceso en ejecución



# Tabla de Archivos

1B V	Nombre del Archivo (14B)	Tamaño (4B)	Dir virtual (4B)	Entradas de archivos 23B x 5

- Cada entrada es de tamaño 23B
- Almacena información sobre archivos de un proceso
- Byte de validez puede tomar valores 0x00 y 0x01
- Tamaño máximo de un archivo es de 64 MB
- Dir Virtuales un valor entre 0 y  $2^{(20)} * 128 - 1$ . Se utilizan 12 bits para el VPN y 15 bits para el offset.
- Los tamaños y direcciones virtuales de los archivos nos permiten saber el estado de la memoria virtual del proceso

# Tabla de Páginas

## Primer Orden



# Tabla de Páginas de Primer Orden

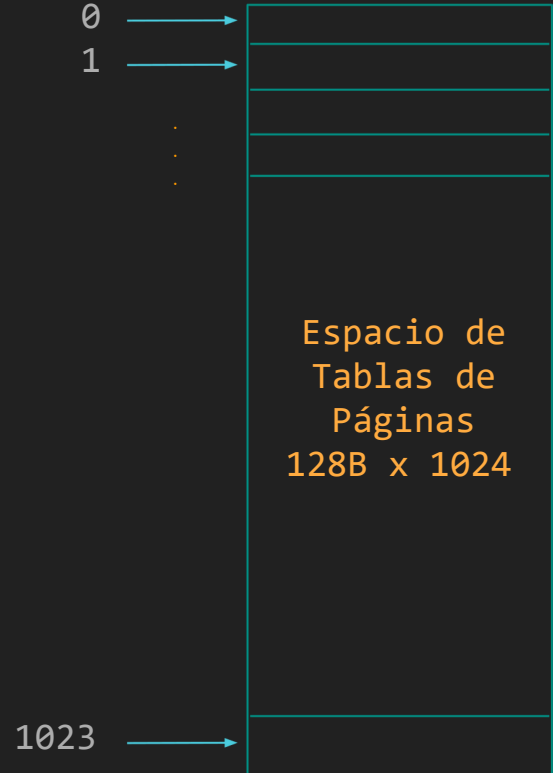
- Se forma a partir de los 6 primeros bits del VPN, por lo que existen 64 ( $2^6$ ) entradas
- Cada entrada tiene un tamaño 2B
- Cada entrada almacenará un número entre 0 y 1023 el cual corresponde a un número de una tabla de páginas de segundo orden

Tabla de  
páginas  
2B x 64

# Tabla de Páginas Segundo Orden

# Espacio de Tablas de Páginas

- Contiene 1024 tablas de páginas de segundo orden.
- Cada una de estas tablas de páginas puede ser enumerada del 0 al 1023.



# Tabla de Páginas de Segundo Orden

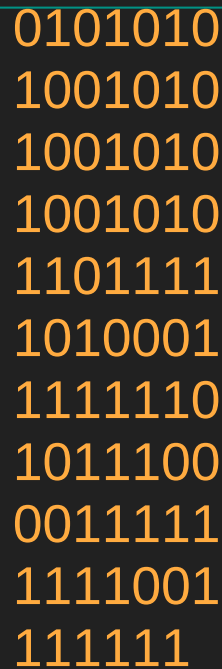
- Se forma a partir de los 6 últimos bits del VPN, por lo que existen 64 ( $2^6$ ) entradas
- Cada entrada tiene un tamaño 2B
- Cada entrada almacenará un número entre 0 y 65536 el cual corresponde al PFN que está asociada la página VPN.

Tabla de  
páginas  
2B x 64

# Bitmap TPS0

- Cada bit del bitmap representa el estado de una tabla de páginas de segundo orden.

128B

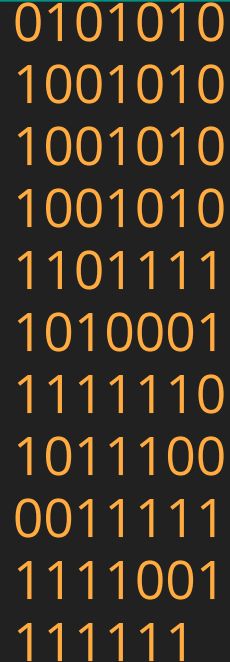


```
0101010
1001010
1001010
1001010
1101111
1010001
1111110
1011100
0011111
1111001
1111111
```

# Frame bitmap

- Cada bit del bitmap representa un
- frame. Refleja el estado de la memoria física.

8KB



0101010
1001010
1001010
1001010
1101111
1010001
1111110
1011100
0011111
1111001
1111111

Memoria virtual

# Transformación de Direcciones Virtuales a Físicas



# Transformación de direcciones

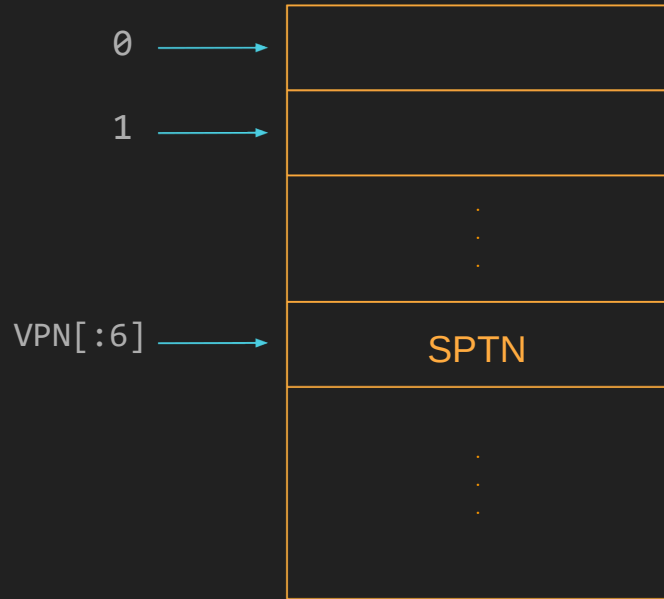
Supongamos que queremos leer archivo con nombre X del proceso con PID Y.

1- Buscamos en la tabla de PCBs un proceso válido cuyo PID sea igual a Y

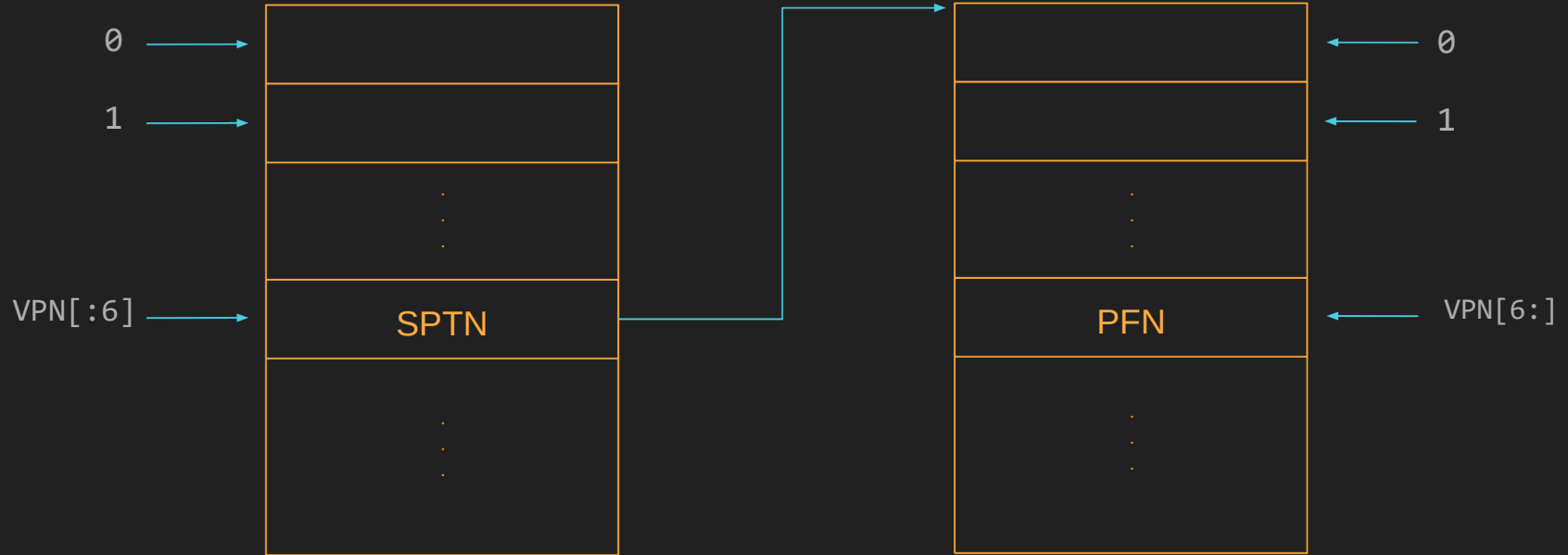
2- Vamos a la tabla de archivos de este proceso, buscamos un archivo que tenga nombre X y obtenemos su dirección virtual.

3- Extraemos el VPN y offset de la dirección virtual.

T. Páginas Primer Orden

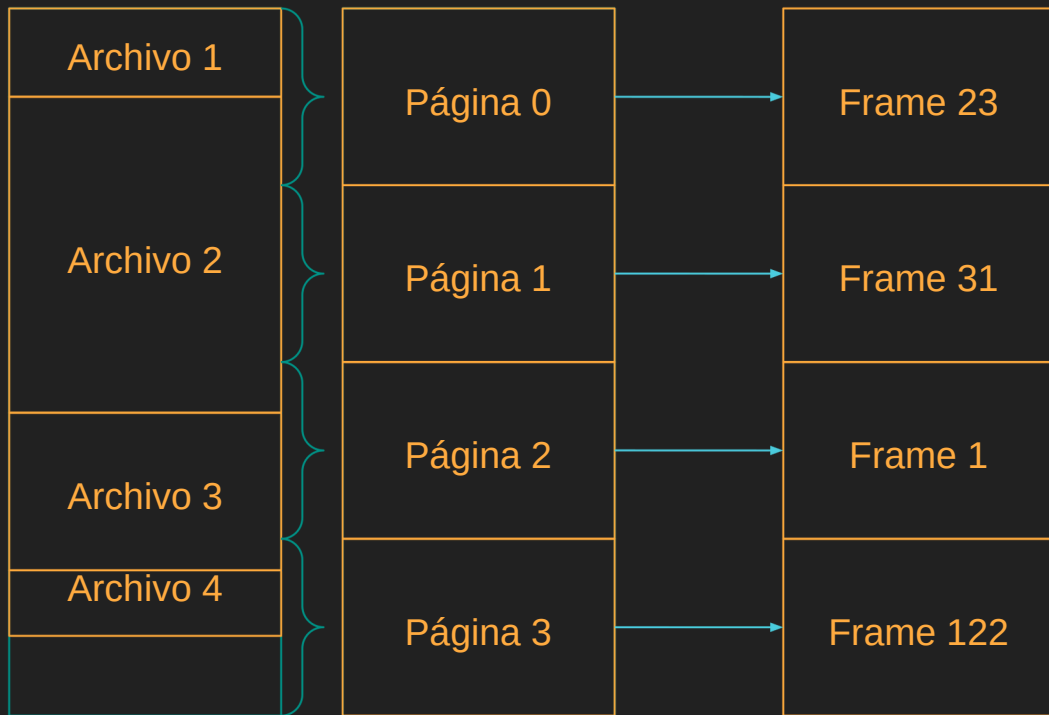


T. Páginas Segundo Orden



Finalmente, la dirección física será igual a PFN seguido del offset.

Leer archivo



Páginas y Frames de 32KB.

Queremos leer el archivo 2 que cuenta con:

- Dir virtual = 0x0000 4000
- Tamaño = 54KB

Pero el archivo utiliza 3 páginas, por lo que tendremos que tomar información de 3 Frames.

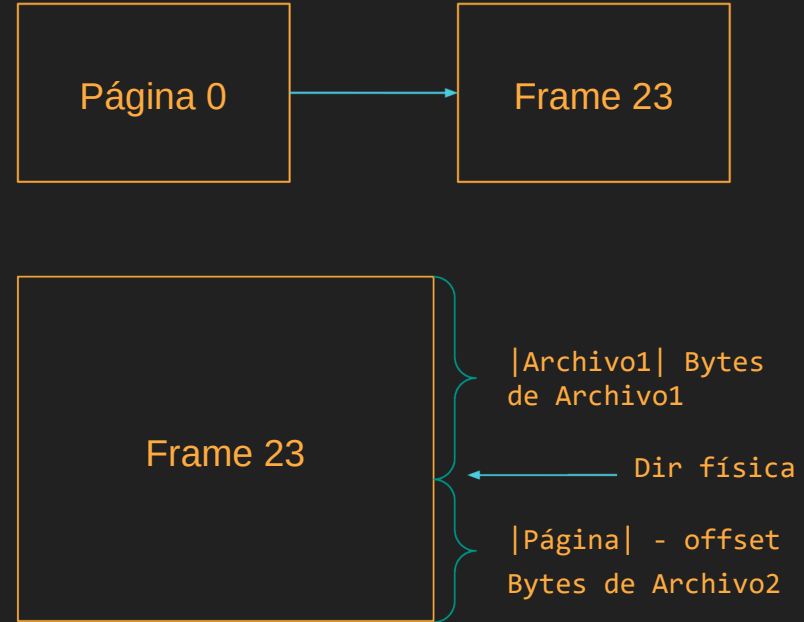
# Lectura - Primera página

1- Obtener el VPN y offset desde la dirección virtual (0x0000 4000):

- VPN = 0
- Offset = 0x0000 4000

2- Obtener el PFN (23) y luego la dirección física.

3- Leer  $2^{15}$  - offset Bytes desde la dirección física, o en otras palabras, |Frame| - offset.



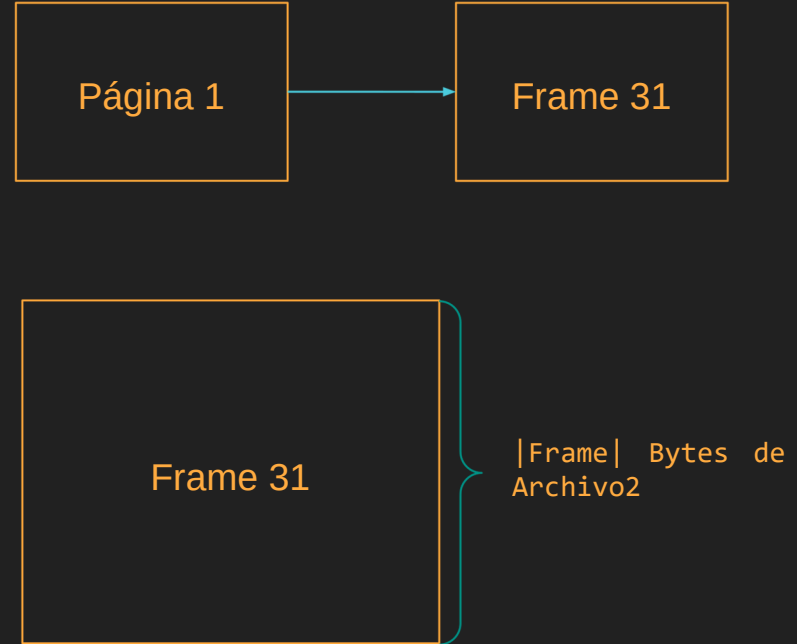
# Lectura - Páginas intermedias

1- Pasamos a la siguiente página, es decir,  $VPN = VPN + 1$  y obtenemos su PFN respectivo (31).

2- Sabemos que la información del archivo se encuentra al inicio de la página, por lo que sabemos que el offset es igual a 0.

3- Con el PFN (31) más el offset(0) podemos obtener la dirección física.

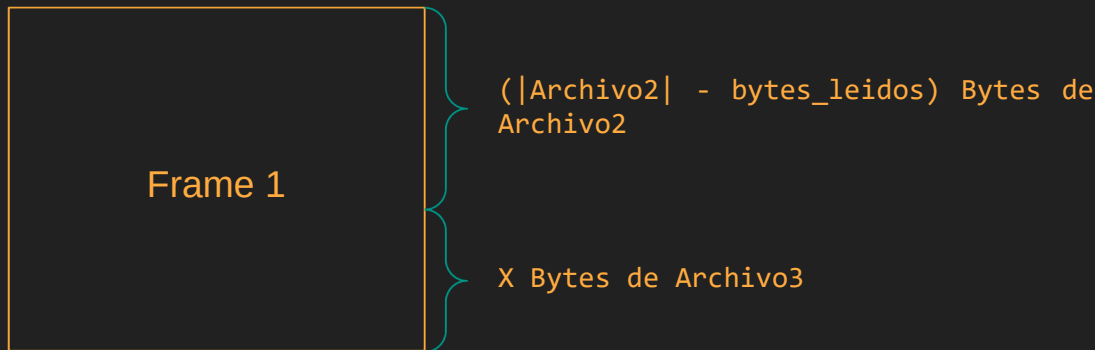
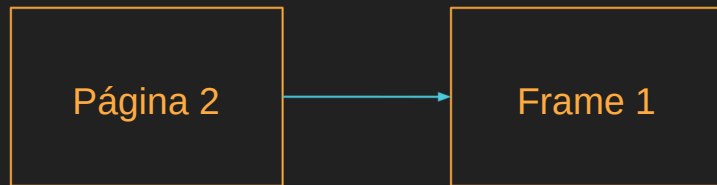
4- Por último, leemos desde la dirección física obtenida 32KB ( $|Frame|$ ).



# Lectura - Página final

La única diferencia con lo anterior es ahora solo se deben leer una cantidad de Bytes igual al tamaño del archivo menos la cantidad de Bytes leídos:

`|Archivo2| - bytes_leídos`



Escribir archivo



# Memoria virtual - escribir archivo

Queremos agregar un archivo

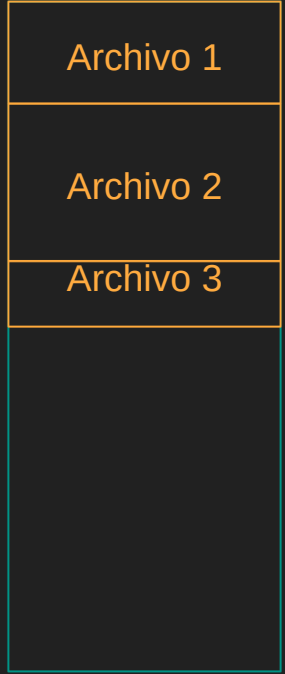
¿Cómo lo hacemos?

128MB

Archivo 1

Archivo 2

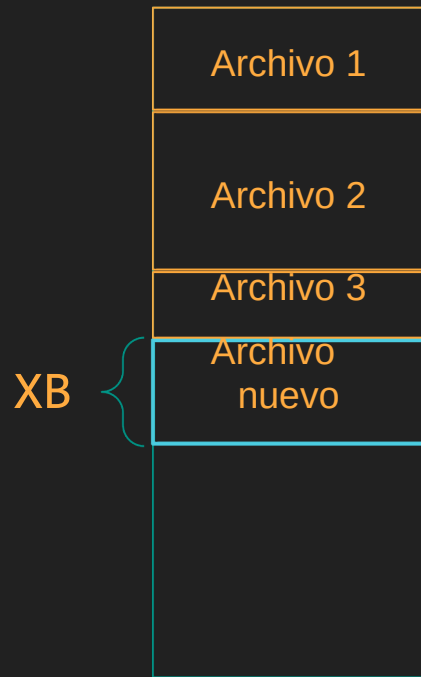
Archivo 3



# Memoria virtual - escribir archivo

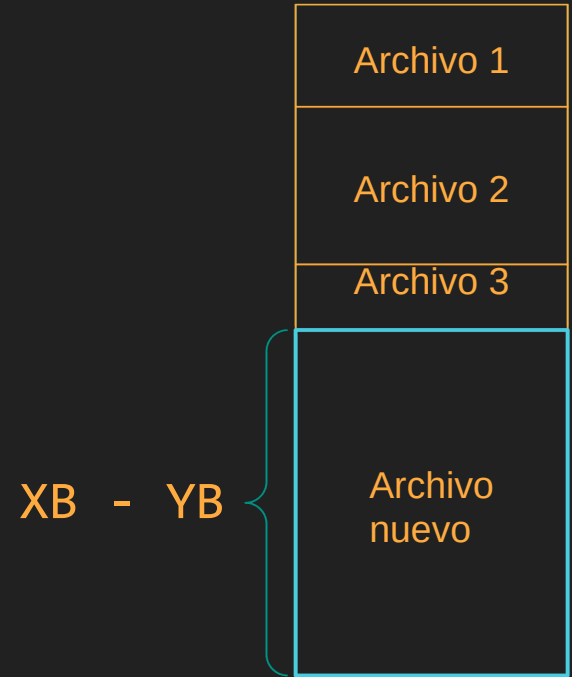
Supongamos que queremos escribir XB.  
Comenzamos a escribir desde el primer espacio disponible.

¿Qué pasa si los XB superan el espacio disponible?



# Memoria virtual - escribir archivo

Dejamos de escribir, por lo que el archivo tendría tamaño  $XB - YB$ , donde  $YB$  son la cantidad de Bytes que no pudieron escribirse.



¿Cómo mostramos todo esto?

¡Con la API!

Manejo de bits

# Leer bit

Se desea tomar el valor del bit de la posición `x` de un `Byte`, para esto se puede realizar:

$$(\text{Byte} \gg x) \text{ AND } 0x01$$

Ejemplo:

Queremos obtener el bit de la posición `3` del `Byte 0b01011101`:

$$\begin{aligned} 0b01011101 \gg 3 &= 0b0001011 \\ 0b0001011 \text{ AND } 0b00000001 &= 0b00000001 \end{aligned}$$

# Reemplazar bit por 1

Se desea reemplazar el valor del bit de la posición **x** de un **Byte** por un 1, para esto se puede realizar:

**Byte OR (0x01 << x)**

Ejemplo:

Queremos reemplazar por un 1 la posición el bit de la posición **4** del Byte **0b10000101**:

**0b00000001 << 4 = 0b00010000**

**0b00010000 OR 0b10000101 = 0b10010101**



# Reemplazar bit por 0

Se desea reemplazar el valor del bit de la posición **x** de un **Byte** por un 0, para esto se puede realizar:

**Byte AND (NOT (0x01 << x))**

Ejemplo:

Queremos reemplazar por un 0 el bit de la posición 5 del Byte

**0b00111101:**

**NOT (0b00000001 << 5) = 0b11011111**

**0b00111101 AND 0b11011111 = 0b00011111**

# Manejo de Bytes en C

# Manejo de Bytes

- `fopen` en modo `r+b`.
- `fseek` con constantes `SEEK_SET`, `SEEK_END`,
- `SEEK_CUR`. `fwrite` y `fread` para leer y escribir
- bytes.

Manejo numérico de bits con shifts.

# Memfilled.bin

Proceso: 91

Nombre proceso: nani

Proceso: 210

Nombre proceso: runedelta

Proceso: 228

Nombre proceso: fifoss

Proceso: 117

Nombre proceso: ssoo

Proceso: 105

Nombre proceso: what

Proceso: 139

Nombre proceso: redes

Proceso: 162

Nombre proceso: cats

Proceso: 234

Nombre proceso: main

# Memfilled.bin: os\_ls\_files

Proceso: 91

knowledg.jpg 110850 Bytes  
mercedes.mp4 1159712 Bytes  
nightcal.mp4 1960382 Bytes  
pepeloni.txt 684 Bytes  
popcorn.mkv 1136865 Bytes

Proceso: 228

amogus.mp4 9658010 Bytes  
caramel.wav 22067920 Bytes  
drums.mp4 9781075 Bytes  
grub.mp4 13400930 Bytes  
yocuando.mp4 1333050 Bytes

Proceso: 117

dino.jpg 47246 Bytes  
facebook.png 1377863 Bytes  
pointer.mp4 616238 Bytes  
smile.png 57160 Bytes  
word.png 1237371 Bytes

Proceso: 105

a.mp4 464107 Bytes  
carrete.mp4 1433810 Bytes  
pepa.mp4 1941103 Bytes  
tom.mp4 5993592 Bytes  
what.mp4 14532987 Bytes

# Memfilled.bin: os\_ls\_files

Proceso: 139

aaaaa.gif 486004 Bytes

chest.wav 302508 Bytes

demo\_new.wav 52528072 Bytes

Midterm.pdf 31227 Bytes

wantisu.mp4 8479972 Bytes

Proceso: 162

greatcat.mp4 299535 Bytes

hecomes.mp4 542939 Bytes

woaeo.mp4 235274 Bytes

Proceso: 210

day.png 3588 Bytes

theme.wav 12567316 Bytes

Proceso: 234

im\_a\_mp3.bin 9513795 Bytes

message.txt 16 Bytes

secret.txt 49474 Bytes

test.jpg 89216 Bytes

# Algunas Referencias:

- [Binary Operators](#)
- [Cambiar un bit de un byte](#)