

SISTEMAS OPERATIVOS



T2: MEMORIA

→ SEGMENTACIÓN

EXERCICIO PÁX 49 DE TEORÍA (con solución)

- Consideremos un sistema con segmentación con las siguientes características
 - las direcciones son de 16 bits (4 bits para el número de segmento y 12 bits para el desplazamiento)
 - cada entrada en la tabla de segmentos tiene 28 bits, los 12 más significativos para el límite y los 16 menos significativos para la base
 - Un proceso tiene dos segmentos y las dos primeras entradas de su tabla de segmentos en memoria valen 0x2EE0400 y 0x79E2020
- a) ¿A donde accede una referencia a la dirección 0x12F0?
- b) ¿A donde accede una referencia a la dirección 0x0342?
- c) ¿A donde accede una referencia a la dirección 0x021F?
- d) ¿A donde accede una referencia a la dirección 0x190A?

	LÍMITE	BASE
TÁBOA DE SEGMENTOS:		
0	0010 1110 1110 0000 0100 0000 0000	
1	0111 1001 1110 0010 0000 0010 0000	

a) $\begin{array}{l} \text{s} \\ \text{d} \\ \hline 0001 \quad 0010 \quad 1111 \quad 0000 \end{array} \} \text{Desp} > \text{Límite? NO!} \quad \} \text{BASE + DESPLAZAMIENTO} \rightarrow \begin{array}{l} 0010 \quad 0000 \quad 0010 \quad 0000 \\ + \quad 0000 \quad 0010 \quad 1111 \quad 0000 \\ \hline 0010 \quad 0011 \quad 0001 \quad 0000 \end{array}$

0x2310

→ PAXINACIÓN

EXERCICIO PÁX 59 DE TEORÍA (con soluciones)

- Supongamos un sistema con direcciones de 16 bits donde 7 bits corresponden al número de página y 9 al desplazamiento dentro de la página
- El tamaño de página sería de 512 bytes $\rightarrow \text{Tam. pax} = 2^{9 \text{ bits desp}}$
- Un proceso referencia la dirección 0x095f ($\underbrace{0000 \quad 1001 \quad 0101 \quad 1111}_{\text{dir. base + despl}} \}$)

CALCULA A DIRECCIÓN DE MEMORIA FÍSICA

* $\begin{array}{r} 1010 \quad 1110 \quad 0000 \quad 0000 \\ + \quad 0000 \quad 0001 \quad 0101 \quad 1111 \\ \hline 1010 \quad 1111 \quad 0101 \quad 1111 \end{array} \} \text{Sumamos dir. base + despl}$

0xAF5F

Teríamos que ver que dir. base hai na entrada nº 4 da táboa de pax. Imaginemos que é a dir $0xAE00 = \underbrace{1010 \quad 1110 \quad 0000 \quad 0000}_{(*)}$

TGR 1

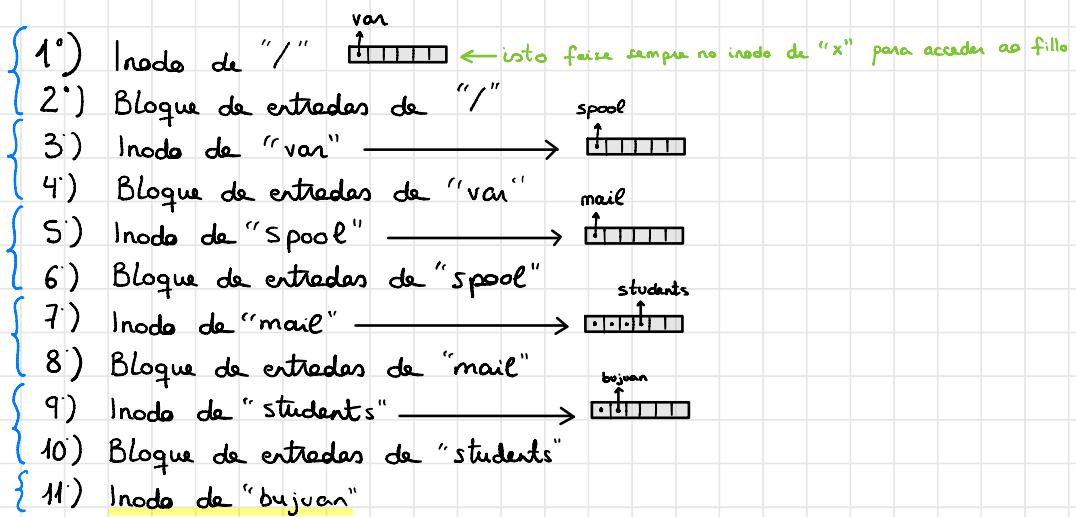
FILE SYSTEMS 1

- 1) In a file system (system V type) the file **bujuan** corresponds to the mailbox of the user **juan**, who is its owner, and the absolute file path is: **/var/spool/mail/students/bujuan**. Answer the following questions:

- 1.1 What is the **minimum number of disks accesses necessary** when running the following open system call?:

`open("/var/spool/mail/students/bujuan", O_RDONLY)`

to obtain the inode of **bujuan**? Directory entries for the different subdirectories are always located in the **first block** of their parent directories, except **students**, whose entry is in the **fourth block** and the file **bujuan**, whose entry is in the **second block**. It is supposed that the Buffer Cache and the Inode Cache are initially empty.



$$\rightarrow \text{Accesos a inodos} = 6 \equiv 1 \text{ acceso lista de inodos}$$
$$\rightarrow \text{Accesos a bloques} = 9$$

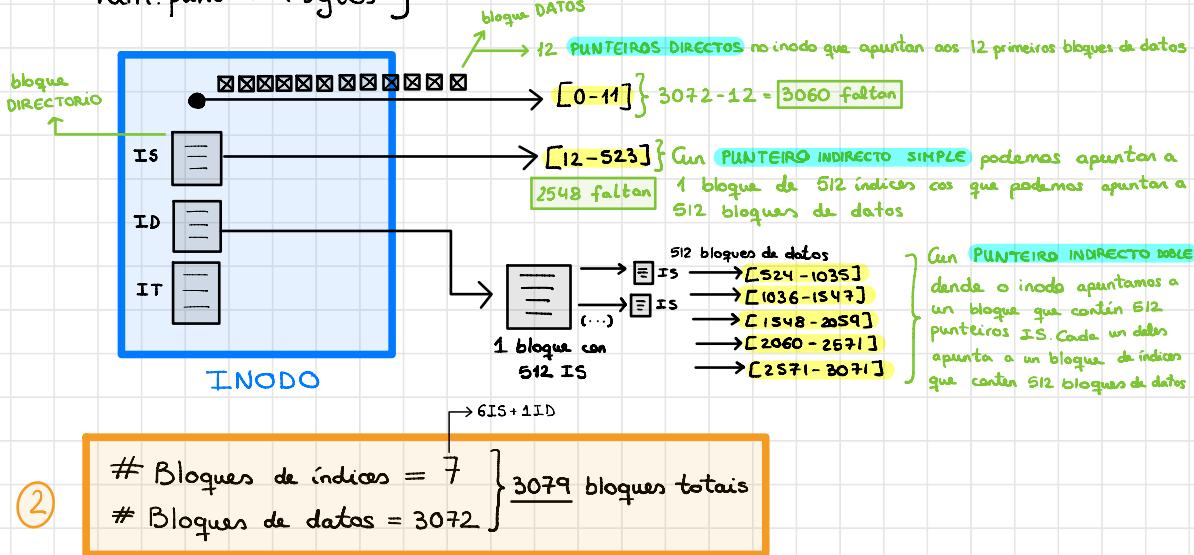
Nº min accesos = 10

- 1.2 In such a file system, the **block size** is 2Kbytes and the **inodes have 12 block direct addresses, one single indirect address, one double indirect address and one triple indirect address**. Moreover, the **block addresses** are represented with 4 bytes.¹ How many disk blocks are necessary for storing the file **bujuan** if its size is 6MBytes? The answer must detail how many blocks are for data and how many for indexes.²

$$1 \quad \underline{6\text{Mb} \cdot 1024\text{ Kb}} / 2\text{ kb} = 3072 \text{ bloques de datos que necesitamos direccionar}$$

(pero los bloques de índices también ocupan)

$$\begin{aligned} \text{Tam. bloque} &= 2 \text{ Kb} \\ \text{Tam. punt} &= 4 \text{ bytes} \end{aligned} \quad \left\{ \frac{2 \text{ Kb} \cdot 1024 \text{ bytes}}{4 \text{ bytes}} = \underline{\underline{512 \text{ punt/bloque}}} \right.$$



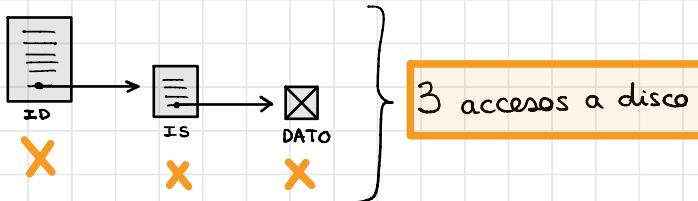
1.3 Following 1.1, once the file is open, the process runs the system call:

`lseek(fd, 4194304, SEEK_SET)`

How many blocks would have to read the operating system to fulfill the sentence:
 $c=fgetc(fd)$?

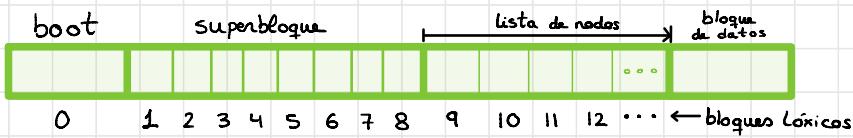
if it is supposed that the Buffer Cache is empty. (Note: $4194304 = 4 \cdot 2^{20}$)

$$4194304 = 4 \cdot 2^{20} = 2 \cdot 2^{10} \cdot \underbrace{2 \cdot 2^{10}}_{2^4 = 2048 = 2 \text{ Kb} = 1 \text{ bloque}} \rightarrow \begin{array}{l} \text{bloque que buscamos} \\ \boxed{\text{bloque 2048 pertenece a } [1548-2059]} \end{array}$$



1.4 What is the logical block number in the file system which corresponds to the root directory inode? (the logical blocks are numbered starting with logical block 0) and what is the logical block number corresponding to the bujuan file inode?, if the following is supposed:

- The inode number of "/" is 2, and to the file bujuan is assigned inode number 35 (the inodes are numbered starting with inode 1).
- The inode size is 128 bytes.
- The boot occupies 1 block and the superblock requires 8 blocks.



→ Cada bloque contiene 2048 bytes

→ Cada nodo ocupa 128 bytes

$$1 \text{ bloque contiene } 2048 \text{ bytes} \cdot \frac{1 \text{ nodo}}{128 \text{ bytes}} = 16 \text{ inodos}$$

1	2	3	4	17	18	19	20	33	34	35	36
5	6	7	8	21	22	23	24	37	38	39	40
9	10	11	12	25	26	27	28	41	42	43	44
13	14	15	16	29	30	31	32	45	46	47	48

9

10

11

...

"/" está en bloque lógico 9

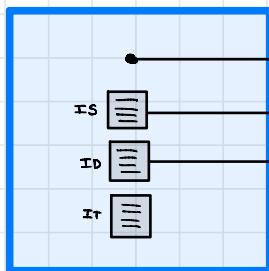
"bujuan" está en bloque lógico 11

TGR 2 FILE SYSTEMS 2

- 1) Un sistema de archivos tipo UNIX tiene un tamaño de bloque de 2Kb, i-nodos con 12 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 4 bytes. ¿Qué bloques son necesarios para representar un archivo de 2.5Mb?

$$\text{bloques} = \frac{2.5 \cdot 1024}{2} = 1028 \text{ bloques de datos}$$

$$\left. \begin{array}{l} \text{Tam. bloque} = 2\text{Kb} \cdot 1024 = 2048 \text{ bytes} \\ \text{Tam. puntero} = 4 \text{ bytes} \end{array} \right\} 2048 / 4 = 512 \text{ punteros/bloque}$$



INODO

$$\begin{aligned} \# \text{ Bloques de índices} &= 1\text{ID} + 3\text{IS} = 4 \\ \# \text{ Bloques de datos} &= 1280 \\ \text{TOTAL BLOQUES} &= 1284 \end{aligned}$$

- 2) Determinar el número de accesos físicos a disco necesarios, como mínimo, en un sistema UNIX, para ejecutar la siguiente operación: $fd = open ("so/practicas/p1/practica1.c", RD_ONLY)$; suponer que la caché del sistema de archivos está inicialmente vacía y que el inodo de “.” está ya en memoria.

LER:

1') Inodo de “.” } Xa en memoria

...	50	i
...		

2') Bloque con entradas de “.”

...	practicas	i
...		

3') Inodo de “so”

...	practicas	i
...		

4') Bloque con entradas de “practicas”

...	p1	i
...		

5') Inodo de “practicas”

...	practica.c	i
...		

6') Bloque con entradas de “practicas”

Asumindo o mellor caso todos os accesos a estes inodos son no mesmo bloque → 1 acceso

* Accesos bloque inodo = 1

* Accesos bloque datos = 4

ACCESES TOTAIS = 5

- 3) En un sistema de archivos UNIX con un tamaño de i-nodo de 64 bytes, un tamaño de bloque de 2048 bytes y en el que la zona de inodos ocupa 2048 bloques, ¿cuántos bloques ocupa un mapa de bits para representar i-nodos libres y ocupados?

Tam. i-nodo = 64 bytes

Tam. bloque = 2048 bytes

Lista inodos = 2048 bloques

Nº INODOS NA LISTA:

$$2048 \text{ bloques} \cdot \frac{2048 \text{ bytes}}{1 \text{ bloque}} \cdot \frac{1 \text{ i-nodo}}{64 \text{ bytes}} = 2^{16} \text{ i-nodos} = 2^{16} \text{ bits no bitmap}$$

Nº BLOQUES MAPA DE BITS:

$$2^{16} \text{ bits} \cdot \frac{1 \text{ byte}}{8 \text{ bits}} \cdot \frac{1 \text{ bloque}}{2048 \text{ bytes}} = 4 \text{ bloques}$$



- 4) En el siguiente código, añadir el código de una función "redireccionar_errores", para que los mensajes de error generados con la función de librería perror (que envía la información al dispositivo de salida de error estándar) se guarden en disco, en el directorio de trabajo actual, en un fichero con nombre "registro_errores".

En todas las líneas de código indicar si existe una llamada a una función de librería de C o bien una llamada al SO:

```
#include <stdio.h>
#include <fcntl.h>

main (int argc, char *argv[]) {
    int fd;
    redireccionar_errores ();
    fd = open(argv[1], O_RDONLY); // open es una llamada al SO
    if (fd == -1) {
        perror("\nerror en open "); // perror es una función de librería de C
        exit(1); // exit es una llamada al SO
    }
}
```

SOLUCIÓN

redireccionar_errores () {

int fd = open("registro_errores", O_WRONLY | O_APPEND); // open es una llamada al SO

close(2); // close es una llamada al SO

dup(fd); // dup es una llamada al SO

}

TGR3

MEMORIA

1. En un sistema la memoria tiene un tiempo de acceso de 50 ns, el TLB de 5 ns, y el disco utilizado para paginación tiene un tiempo medio de búsqueda de 3 ms, una latencia de 1ms y una velocidad de transferencia de 4 Mbytes/seg. Sabiendo que el tamaño de página es de 4K, calcular el tiempo de acceso efectivo en cada uno de los siguientes casos:

- a) Sin paginación
 - b) Con paginación y sin TLB
 - c) Con paginación y una probabilidad de acierto en el TLB del 80%
 - d) Con paginación, una probabilidad de acierto en el TLB del 100% y una probabilidad de fallo de página de 10^{-6}
- ↳ que ronda

$$TAE = PA + PF = (\underset{\substack{(tempo\ acceso\ efectivo)}}{p \cdot ta}) + (\underset{\substack{(prob.\ acierto)}}{1-p}) \cdot (\underset{\substack{(prob.\ fallo)}}{tf})$$

↓ ↓
tempo mejor escenario tempo peor escenario

$$\begin{aligned} T_{acceso} &= 50\text{ns} \\ T_{TLB} &= 5\text{ns} \\ T_{busq\ disco} &= 3\text{ms} \\ LATENCIA &= 1\text{ms} \\ Vel.\ transf &= 4\text{Mbytes / seg} \\ Tam.\ pax &= 4\text{Kbytes} \end{aligned}$$

a) $TAE = 50\text{ns}$

b) $TAE = 50\text{ns} + 50\text{ns} ; TAE = 100\text{ns}$

c) Mejor escenario: Acceso TLB + Acceso MEMORIA = 55ns
 $(5) \quad (50)$

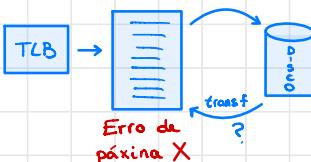
Pior escenario: Acceso TLB + Acceso MEMORIA + Búsqueda MEMORIA = 105ns
 $(5) \quad (50) \quad (50)$

↓
Producirse un ERRO

↓
Como o TLB NON lle dan
o índice ten que buscar un a
un alta o final

$$TAE = 0'8 \cdot 55 + 0'2 \cdot 105 = 65\text{ns}$$

- d) Producirse un error de página (ignoramos o error de TLB)



$$\begin{aligned} tf &= t_{busq\ disco} + t_{latencia} + t_{latencia} = \\ &= 3\text{ms} + 1\text{ms} + 1\text{ms} = 5\text{ms} \end{aligned}$$

↓
4Mb → 1 seg
4096Kb → 2 seg
4 Kb → 2 seg } 1ms

$$TAE = 0'999999 \cdot 55\text{ ns} + 10^{-6} \cdot 5 \cdot 10^6 \text{ ns} = 60\text{ ns}$$

2. A continuación se muestra la salida del comando pmap para un proceso

```
#) pmap 5396
5396: ./a.out
0000000000400000      56K r-x-- a.out
000000000060e000      4K rw--- a.out
000000000060f000      16K rw--- [ anon ]
0000000000130d000     132K rw--- [ anon ]
00007f933f9b6000     1140K ----- file.tar
00007f933fad3000 500000K rw-s- [ shmid=0x45e0022 ]
00007f935e31b000 500000K rw-s- [ shmid=0x45e0022 ]
00007f937cb63000 2441408K rw--- [ anon ]
00007f9411b93000 1660K r-x-- libc-2.19.so
00007f9411d32000 2048K ----- libc-2.19.so
00007f9411f32000 16K r---- libc-2.19.so
00007f9411f36000 8K rw--- libc-2.19.so
00007f9411f38000 16K rw--- [ anon ]
00007f9411f3c000 128K r-x-- ld-2.19.so
00007f9412000000 1140K ----- file.tar
00007f941211d000 12K rw--- [ anon ]
00007f9412140000 24K rw-s- [ shmid=0x4608024 ]
00007f9412146000 24K rw-s- [ shmid=0x45f0023 ]
00007f941214c000 24K rw-s- [ shmid=0x4608024 ]
00007f9412152000 24K rw-s- [ shmid=0x45f0023 ]
00007f9412158000 16K rw--- [ anon ]
00007f941215c000 4K r---- ld-2.19.so
00007f941215d000 4K rw--- ld-2.19.so
00007f941215e000 4K rw--- [ anon ]
00007ffcb7251000 132K rw--- [ stack ]
00007ffcb72dd000 8K r-x-- [ anon ]
00007ffcb72df000 8K r---- [ anon ]
ffffffffff600000 4K r-x-- [ anon ]
total            3448060K
```

Shmid = rexión memoria compartida

- a) ¿Se trata de un sistema de 32 bits o 64 bits?
- b) ¿Se ha compilado estáticamente?
- c) ¿Cuanto ocupa el proceso en memoria?
- d) ¿Qué hay en la dirección de memoria física 0x7f9412000000?
- e) ¿Utiliza alguna región de memoria compartida? ¿cuántas?

a) 64 bits, pq todas as direcciones están compostas por 16 bits en hexadecimal

b) Non, pq vense os segmentos de LIBC e LD

→ Interpretar como "librería dinámica"

c) Non o podemos saber porque pmap mostra o ESPAZO VIRTUAL

d) Non o podemos saber porque pmap mostra o ESPAZO VIRTUAL

c) Si. Usa TRES rexións distintas $\left\{ \begin{array}{l} 0x4608024 \\ 0x45f0023 \\ 0x45e0022 \end{array} \right\}$ mapeadas tres veces cada unha

3. Un sistema de 16 bits tiene páginas de 1 Kbytes y 32K de memoria instalada. Cada entrada de la tabla de páginas consta de 16 bits, cuyo significado es el siguiente

- bits 0-5: Número de marco físico, (6 bits más significativos de la dirección de memoria física)
- bit 6: Protección: 1, (r/w), 0 (ro)
- bit 7: Privilegio (1, página solo accesible en modo kernel)
- bit 8: Presencia (1, página en memoria)
- bits 9: Referencia (1, página referenciada)
- bits 10-15: 000000, entrada no válida; 111111, entrada válida

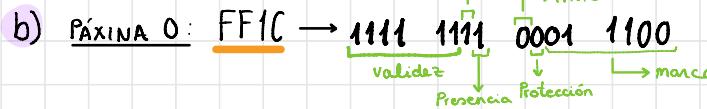
En dicho sistema en un instante dado hay, aparte del S.O., un proceso, P_1 , cuya tabla de páginas se muestra a continuación

	0	1	2	3	4	5	6	7	...
P_1	FF1C	FF1D	FF1E	FC40	FC4F	FC4E	FF46	0000	...

	...	38	39	3A	3B	3C	3D	3E	3F
P_1	...	0000	FC50	FF4C	FF50	FF80	FF81	FFC2	FFC3

- ¿Tiene dicho sistema memoria virtual?
- Muéstrese cómo está dicho proceso en memoria
- ¿Cuánta memoria libre queda en el sistema?
- ¿Cuál es el tamaño virtual del proceso P_1 ? ¿Cuánto ocupa en memoria?
- ¿En qué dirección virtual comienzan los datos y la pila del proceso?
- ¿En qué dirección virtual ve el proceso el S.O.? ¿Dónde está el S.O. en la memoria física?
- ¿Qué ocurriría si P_1 intenta leer los contenidos de la posición de memoria 0x04F9? ¿A qué posición de memoria acceden realmente? ¿Y si intentan escribir en dicha posición?
- ¿Qué ocurriría si P_1 intentan leer los contenidos de la posición de memoria 0xE42A? ¿A qué posición de memoria se accede realmente?
- ¿Podría implementarse LRU en este sistema?
- ¿Podría P_1 tener declarada una variable local de `main char a[3300]`? ¿Y si fuese externa?

a) Hai bit de presencia, polo que si ten memoria virtual



Entrada válida co bit de presencia 1 polo que si está en memoria. Entrada referenciada con permiso de só lectura e modo usuario. A páxina está na dir. física

$$\begin{array}{cccccc} 0111 & 0000 & 0000 & 0000 \\ \text{marco} & & & & & \end{array} = 0x7000$$

VALIDEZ	→ 1 = válida → 0 = non válida
REFERENCIA	→ 1 = referenciada → 0 = non referenciada
PRESENCIA	→ 1 = presente → 0 = non presente
PRIVILEXIO	→ 0 = user → 1 = kernel
PROTECCIÓN	→ 0 = lectura → 1 = lectura/escritura
MARCO	Obter dirección

PÁXINA 1: FF1D → 1111 1111 0001 1101

Entrada válida, referenciada e presente en memoria. Está en modo usuario e con permiso de só lectura. A súa dir da memoria física é 0111 0100 0000 0000 → 0x7400

PÁXINA 2: FF1E → 1111 1111 0001 1110

Entrada válida, referenciada e presente en memoria. Está en modo usuario con permiso de só lectura e a súa dir da memoria física é 0111 1000 0000 0000 → 0x7800

(...)

PÁXINA 3C: FF80 → 1111 1111 1000 0000

Páxina válida, referenciada e presente en memoria. Está en modo kernel con permiso só de lectura. O código do SO está en 0x0000

Se estivera en modo LECTURA / ESCRITURA
sería DATOS

(...)

c) Se facemos o apartado anterior entero vemos que quedan 10 marcos de 1Kbytes ocupados e 22 marcos libres, polo que hai 22Kbytes libres

4. ¿Por que se utilizan tablas de páginas en varios niveles?

En espazos virtuais moi grandes, non é necesario asignar espazo para tábors de páxinas nos ocos do espacio de direccións, empregando así menos memoria para os tábors de páxina que no caso de usa TP dun nivel.

5. ¿Qué es la anomalía de Belady y que tipo de algoritmos de reemplazo no la presentan

Situación que se pode presentar nalgúns algoritmos de reemplazo de páxina, con algunas cadeas de referencia concretas, na que ao aumentar o nº de marcos asignados aumenta o número de fallas de páxina. FIFO presenta dita anomalía, LRU e óptimo garantizan que non. Os algoritmos que NON a presentan son os denominados ALGORÍTMOS DE PILA, nos cales o consumo residente con N marcos é un consumo residente con $N+1$ marcos.

6. Considérese las siguiente cadena de referencias a memoria (donde cada letra representa una página) ABCDAABBCCDDDDDEFGEFGEFGEF-GAAAD. Indíquese cuantos fallos de páxina hay y muéstrese el conjunto residente con cada referencia utilizando

→ Se nos 3 últimos tempos non se referencia á páxina X, sácase de memoria.

a) Working Set de ventana 4. Número de fallos de página: 12

b) L.R.U. con 4 marcos. Número de fallos de página: 9

a)

A	B	C	D	A	A	B	B	C	C	D	D	D	D	E	F	G	E	F	G	A	A	A	D	//////
*	*	*	*			*	*	*						*	*	R							*	*
A	A	A	A	A	B	B	B	C	C	D	D	D	D	E	F	G	E	F	G	A	A	A	D	
B	B	B	B	=	D	A	A	B	B	C	C	C	C	=	D	E	F	E	G	=	=	=	=	
C	C	C	D	C	D	D	A	A	B	B	B	B	C	=	D	E	F	E	F	G	G	G	F	

b)

A	B	C	D	A	A	B	B	C	C	D	D	D	D	E	F	G	E	F	G	A	A	A	D	//////
*	K	*	*											*	*	*								*
A	B	C	D											E	F	G								
A	B	C	=	=	=	=	=	=	=	=	=	=	=	D	E	F	=	=	=	=	=	=		
A	B	A	B	=	=	=	=	=	=	=	=	=	=	C	D	E	=	=	=	=	=	=		

7. Un sistema de 16 bits tiene paginación con memoria virtual. El tamaño de página es de 1K y cada entrada de la tabla de páginas tiene 8 bits, cuyo significado es el siguiente

- **bits 0-5:** Número de marco físico, (6 bits más significativos de la dirección de memoria física)
- **bit 6:** Protección: 1, (r/w), 0 (ro)
- **bit 7:** Privilegio (1, página solo accesible en modo kernel)
- **bit 8:** Presencia (1, página en memoria)
- **bits 9:** Referencia (1, página referenciada)
- **bits 10-15:** 000000, entrada no válida; 111111, entrada válida

En un instante dado la memoria física del sistema está como se muestra a continuación:

16	0x4000	P_1 STACK2
15	0x3C00	
14	0x3800	P_1 CODE2
13	0x3400	
12	0x3000	P_1 STACK1
11	0x2C00	P_1 DATA2
10	0x2800	
9	0x2400	
8	0x2000	P_1 CODE0
7	0x1C00	P_1 DATA0
6	0x1800	P_1 DATA1
5	0x1400	P_1 DATA3
4	0x1000	
3	0x0C00	S.O. (data)
2	0x0800	S.O. (data)
1	0x0400	S.O. (code)
0	0x0000	S.O. (code)

17 páginas

- a) ¿Cuánta memoria hay instalada en el sistema?
17Kbytes
- b) ¿Cuánta hay libre?
5kbytes (SO ocupa 4Kbytes e P_1 8Kbytes)

- c) Sabiendo que en dicho sistema los procesos ven el S.O. en las direcciones virtuales más altas, y que P_1 tiene 3 páginas de código (CODE0, CODE1 y CODE2) que comienzan en la dirección virtual 0x0000, 4 páginas de datos (DATA0, DATA1, DATA2 y DATA3) que comienzan en la dirección virtual 0x0C00, y 3 páginas de pila (STACK0, STACK1 y STACK2) que comienzan en la dirección virtual 0xE400
Construir la tabla de páginas para dicho proceso en dicho instante

0x0000 → Páginas 0, 1 e 2

0x0C00 → Páginas 3, 4, 5 e 6

0xE400 → Páginas 39, 3A, 3B

$$\text{dende } 0x0000 \text{ a } 0x1000 = 4 \text{ octos. Como E} = 14 \longrightarrow 14 \cdot 5 + 1 = 57 = 11\ 1001 = \boxed{39}$$

0 octo de 0x0400

Entrada Página	Bits 10-15 (Valida)	Bit 9 (R)	Bit 8 (P)	Bit 7 K/U	Bit 6 R/W	Bits 0-5 (n marco) ?	Valor entrada Binario (HEX)
3F (SO 3)	111111	1	1	1	1	3(000011)	1111 1111 1100 0011 (0xFFC3)
3E (SO 2)	111111	1	1	1	1	2(000010)	1111 1111 1100 0010 (0xFFC2)
3D (SO 1)	111111	1	1	1	0	1(000001)	1111 1111 1000 0001 (0xFF81)
3C (SO 0)	111111	1	1	1	0	0(000000)	1111 1111 1000 0000 (0xFF80)
3B (STK2)	111111	1	1	0	1	16(010000)	1111 1111 0101 0000 (0xFF50)
3A (STK1)	111111	1	1	0	1	12(001100)	1111 1111 0100 1100 (0xFF4C)
39 (STK0)	111111	0	0	0	1	(???????)	1111 1100 01?? ????
...	000000	0	0	0	0	00000	0000 0000 0000 0000 (0x0000)
...	000000	0	0	0	0	00000	0000 0000 0000 0000 (0x0000)
.....	000000	0	0	0	0	00000	0000 0000 0000 0000 (0x0000)
6 (DATA3)	111111	1	1	0	1	5(000101)	1111 1111 0100 0101 (0xFF45)
5 (DATA2)	111111	1	1	0	1	11(001011)	1111 1111 0100 1011 (0xFF4B)
4 (DATA1)	111111	1	1	0	1	6(000110)	1111 1111 0100 0110 (0xFF46)
3 (DATA0)	111111	1	1	0	1	7(000111)	1111 1111 0100 0111 (0xFF47)
2 (CODE2)	111111	1	1	0	0	14(001110)	1111 1111 0000 1110 (0xFF0E)
1 (CODE1)	111111	0	0	0	0	???????	1111 1100 00?? ????
0 (CODE0)	111111	1	1	0	0	8(001000)	1111 1111 0000 1000 (0xFF08)

TGR 4

PROCESOS

* NON APROPIATIVOS

$$\text{Tiempo promedio} = \frac{\sum \text{espere } p_i}{\text{num. procesos}}$$

$$\text{Tiempo promedio} = \frac{\sum (\text{espere } p_i + \text{ejecución } p_i)}{\text{num. procesos}}$$

- * **FCFS** → First come first serve (FIFO)
- * **SJF** → Shortest job first
- SRTF** → Shortest remaining time first
- * **Pri (n)** → Prioridades no apropiativas (SEN desaloca)
- Pri (a)** → Prioridades apropiativas (CON desaloca)
- RR** → Round Robin (QUANTUM)

1.- Considerar los siguientes procesos con los tiempos de llegadas, prioridades y ráfagas de CPU.

Proceso	Ráfaga de CPU	Prioridad	Tiempo llegada
A	50 ms	4	0 ms
B	20 ms	1	20 ms
C	100 ms	3	40 ms
D	40 ms	2	60 ms.

Mostrar cómo se planifican estos procesos con los algoritmos FCFS, SJF, SRTF, Prioridades Apropiativo y no Apropiativo (un valor más pequeño en la tabla indica más prioridad) y Round Robin con quantum de 30 ms. ¿Cuál es el tiempo de espera promedio para los distintos algoritmos y esta carga de trabajo?

Llegada	A	B	C	D															
FCFS	A A A A A B B C C C C C C C C C C D D D D																		
SJF	A A A A A B B D D D D C C C C C C C C C C C C																		
SRTF	A A B B A A A D D D D C C C C C C C C C C C C																		
PRI.n	A A A A A B B D D D D C C C C C C C C C C C C																		
PRI.a	A A B B C C D D D D C C C C C C C C A A A A																		
RR(30)	A A A B B A A C C C D D D C C C D C C C C																		
	0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200																		

$$T_{FCFS} = \frac{0+30+30+110}{4}$$

$$T_{SRTF} = \frac{20+0+70+10}{4} = 25 \text{ ms}$$

Mejor caso

$$T_{PRI(a)} = \frac{160+0+40+0}{4}$$

$$T_{SJF} = \frac{0+30+70+10}{4}$$

$$T_{PRI(n)} = \frac{0+30+70+10}{4}$$

$$T_{RR} = \frac{20+10+70+70}{4}$$

2.- En el código que se muestra a continuación y que corresponde al ejecutable “./a.out”. ¿Cuántos procesos se crean? (no se cuenta la creación del primer proceso desde el shell) ¿Qué salida produce por pantalla?

- a) ninguno → Pq non hai ningún fork e non se conta o primeiro. A saída que se produce é “i vale 0” nun bucle infinito.
- b) uno
- c) 500
- d) 499
- e) 100
- f) 99
- g) infinitos

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define MAX 500

main(int argc, char * argv[])
{
    int i=0;

    printf ("i vale %d \n",i);

    i++;

    if (i==MAX)
        exit(0);
    execl("./a.out", "./a.out", NULL);
    i=i+5;
}
```

3.- En el código que se muestra a continuación y que corresponde al ejecutable “./a.out”. ¿Cuántos procesos se crean? (no se cuenta la creación del primer proceso desde el shell). ¿Qué salida produce por pantalla?

- a) ninguno
- b) uno
- c) 500
- d) 499
- e) 100
- f) 99
- g) infinitos

Prodícese a saída “i vale 1” un nº infinito de veces e de cada vez un proceso distinto. En cada iteración créase un proceso fillo e mátase ao pai, así constantemente.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define MAX 500
```

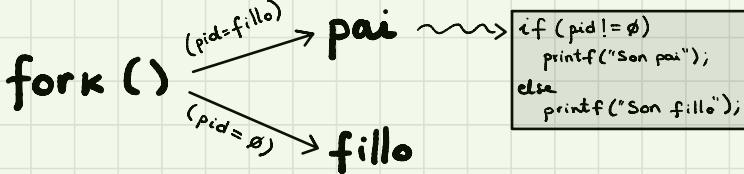
```

main(int argc, char * argv[])
{
    int i=0;
    pid_t pid;

    i++;
    printf ("i vale %d \n",i);

    if ((pid=fork())!=0)
        exit(0); //matamos ao pai
    sleep (1);
    if (i==MAX)
        exit(0);
    execl("./a.out", "./a.out", NULL);
    i=i+5;
}

```



ESCENARIO A → PAI ESPERA (waitpid) } 1º plano

ESCENARIO B → PAI NON ESPERA } 2º plano

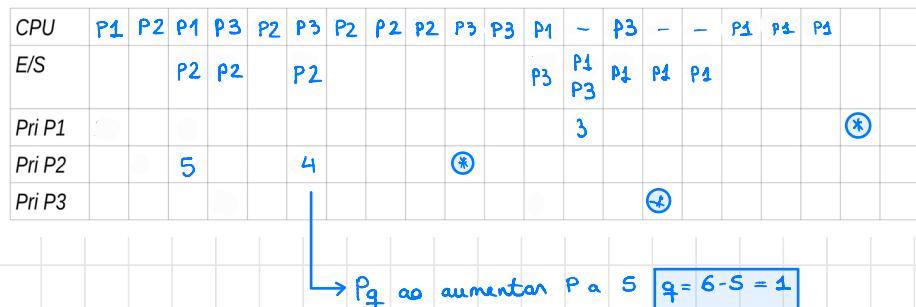
4.- Considérese un sistema con planificación por prioridades dinámicas y apropiativas. El funcionamiento del sistema es el siguiente:

1. El sistema tiene definidas **prioridades de 1 a 5** (menor número indica menor prioridad)
2. **Se ejecuta siempre el proceso de más prioridad.** Cuando dos procesos tienen la misma prioridad
 1. si uno de ellos está en CPU se deja al que está en CPU.
 2. Si ninguno de ellos está en CPU obtiene la CPU el que lleva mas tiempo en la cola de lista.
3. Si un proceso es expulsado de la CPU, cuando la vuelve a obtener su quanto vuelve a comenzar.
4. A cada proceso se le asigna un quanto que depende de su prioridad y es igual **6-P**, siendo P la prioridad (p.e. a un proceso de prioridad 3 se le asigna un quanto de 3).
5. Suponemos que **varios procesos pueden realizar e/s concurrentemente**.
6. El sistema recalcula la **prioridad** del proceso que abandona la CPU de la siguiente manera:
 1. Si el cuanto expira antes que la ráfaga, su prioridad diminuye en 1.

2. Si la ráfaga acaba antes que el cuanto, su prioridad aumenta en 1.
 3. Si el proceso es expulsado de la CPU por otro de más prioridad su prioridad no varía
 4. Dado que esto es un ejemplo y ponemos ráfagas representadas por números enteros (cosa que no ocurre en un sistema real) supondremos que cuando el instante de terminación de la ráfaga coincide con el de terminación del cuánto, es el cuánto lo que termina antes y, por tanto, la prioridad del proceso disminuye

Mostrar cómo sería la planificación para los procesos que se muestran en el cuadro. Las ráfagas e/s se muestran entre paréntesis, así 4(3)4 representa 4 cpu + 3 e/s + 4cpu

Proceso	Ráfagas	Prioridad inicial	Instante de llegada
P1	$Q = 6 - 2 = 4$	3(4)3	2
P2	$Q = 6 - 4 = 2$	1(2)1(1)3	4
P3	$Q = 6 - 3 = 3$	4(2)1	3



5.-Considere los siguientes procesos con tiempos de llegada y ráfagas de CPU

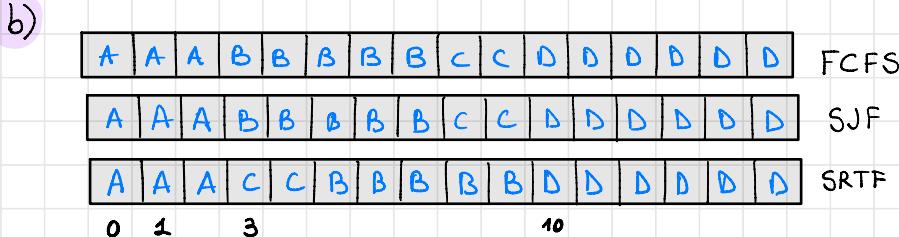
Proceso	Tiempo de Llegada	Tiempo de CPU
A	0	3
B	1	5
C	3	2
D	10	6

a) Se presenta una posible planificación. Dígase de qué planificación se trata, y calcúlese el tiempo de retorno promedio para esta planificación

??? A A A B B B C C B D D D D D D D

b) Con estos procesos llegando en estos instantes. ¿Cuál es el mejor tiempo de espera promedio que puede obtenerse?

a) RR con quantum=4 $\rightarrow T_{retorno} = \frac{3+(7+2)+(4+2)+(0+6)}{4} = 6$



$$T_{FCFS} = T_{SJF} = \frac{0+2+5+0}{4} = 7/4$$

$$T_{SRTF} = \frac{0+4+0+0}{4} = 1 \leftarrow \text{TIEMPO MÁS CURTO}$$

6.- Dos procesos A y B tienen una ráfaga de CPU de 50 ms y un proceso C tiene cuatro ráfagas de CPU de 10 ms. intercaladas con tres ráfagas de E/S a disco de 30 ms. Los procesos tienen el mismo tiempo de llegada con el orden C, A, B. Muestre el diagrama de planificación y calcule el tiempo de retorno promedio y el porcentaje de utilización de la CPU y del disco bajo las políticas de planificación de la CPU RR con quantum 50 ms y SRTF (Shortest Remaining Time First).

A \rightarrow	5
B \rightarrow	5
C \rightarrow	1 (4)
Orde chegada	C-A-B

RR	CPU	C	A	A	A	A	A	B	B	B	B	C		C	
	E/S		C	C	C							C	C	C	C
SR	CPU	C	A	A	A	C	A	A	B	C	B	B	B	B	C
TF	E/S		C	C	C		C	C	C	C	C	C			

RR : $T_{RR} = \frac{(1+5)+(6+5)+20}{3} = \frac{37}{3}$ % CPU = $\frac{14}{20} \cdot 100 = 70\%$

SRTF : $T_{SRTF} = \frac{(1+6)+(7+6)+14}{3} = \frac{34}{3}$ % CPU = $\frac{14}{14} \cdot 100 = 100\%$

7.-Propóngase el código de un programa que liste el directorio que se le pasa como parámetro pero que no liste durante más de 10 segundos. Ayudas: puede usarse una de la llamadas “exec” para ejecutar el programa externo “ls”; la función “sleep” deja un proceso en espera los segundos que se le pasan como parámetro y “kill (pid,SIGTERM)” termina el proceso pid.

```
main (int argc, char * argv[])
{
    /*argumentos que se introducen
     *argumentos en cuestión
    pid_t p;

    if (argv[1]==NULL) {
        printf ("Nada que listar\n");
        return;
    }
    if ((p=fork())==0) { /*proceso hijo*/
        execlp ("ls"."ls", "-l", argv[1], NULL);
        exit (255); /*por si falla execlp */
    }
    sleep (10);
    kill (p, SIGKILL); // mata al fillo
}
```

TGR 5

ENTRADA - SAÍDA

1. FEITO NO PROPIO BOLETÍN

2. Asúmase que tenemos un disco duro con 250 cilindros numerados del 0 al 249, cuya cabeza está sobre el cilindro 100 (atendiendo una petición de acceso previa) y se está moviendo en sentido ascendente ($0 \rightarrow 249$). Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: $\langle 99, 102, 125, 29, 247, 95, 110, 90, 198, 10 \rangle$, indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

FCFS o FIFO:	99	102	125	29	247	95	110	90	198	10
SSTF:	99	102	95	90	110	125	198	247	10	29
SCAN:	102	110	125	198	247	99	95	90	29	10
C-SCAN:	102	110	125	198	247	10	29	90	95	99

3. En un momento dado, la cola de peticiones de E/S para acceder a cilindros de un disco duro contenía las peticiones: $\langle 88, 33, 18, 90 \rangle$. Desconocemos el cilindro que estaba siendo accedido antes de estas peticiones, pero sabemos que dichas peticiones fueron atendidas en el orden siguiente $\langle 88, 33, 18, 90 \rangle$. Indíquese si los siguientes algoritmos pudieron haber sido (o no) usados para la planificación del acceso al disco. Razónese brevemente la respuesta:

algor.	sí/no	breve justificación de la respuesta
FIFO:	Si	Porque as peticions foron atendidas na orde de chegada
SSTF:	Non	Porque non foron atendidas dándolle prioridade as solicitudes máis próximas (menor tempo de búsqueda). Por exemplo despois de 88 debería ir 90, non 33
SCAN:	Si	Inicialmente baixando consecutivamente. Ao baixar atende $\langle 88, 33, 18 \rangle$ e ao subir a 90

8. Si vamos a transferir 1Mbyte a un dispositivo de E/S. ¿Quién generará más interrupciones durante dicha operación?. ¿Un controlador que use E/S programada (polling), otro que use E/S por interrupciones, u otro que use DMA?. Justifique brevemente la respuesta.

El controlador que use Interrupciones, pues genera una interrupción cada vez que un dato es transferido, mientras que el DMA potencialmente podrán originar sólo una interrupción al final de la transferencia. Por su parte polling no genera interrupciones.

9. Tengo un disco de 2 caras, con 80 cilindros y un total de 18 sectores por pista. Cada sector contiene 512 bytes (0.5Kbytes). Si lo he formateado usando bloques de 2Kbytes, ¿Cuántos bloques son accesibles para el Sistema Operativo?

$$2 \text{ caras} \times 80 \text{ cilindros} ==> 160 \text{ pistas}$$

$$160 \text{ pistas} \times 18 \text{ sectores/pista} \times 0.5\text{Kb/sector} \times 0.5 \text{ bloques/KB} = 160 \times 9 \times 0.5 \text{ bloques} = 720 \text{ bloques}$$

DISCO: Conxunto de platos que xiran solidariamente

CARA: Cada superficie dun plato

PISTA: Coroas concéntricas circulares que hai en cada cara

CILINDRO: Conxunto dunha mesma pista en distintas caras

SECTORES: Cada cilindro está formado por sectores ▷

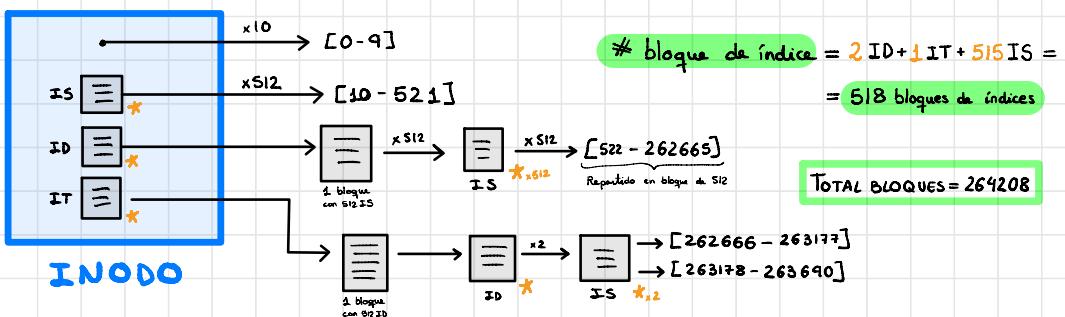
XANEIRO 2021

· PARTE SISTEMAS DE FICHEIROS

- 1 Un sistema de ficheiros tipo UNIX System V ten un tamaño de bloque de 4Kbytes, i-nodos con 10 enderezos directos, un indirecto simple, un indirecto dobre e un indirecto triple. Usa enderezos de bloque de 8 bytes. Calcula cuntos bloques son necesarios na área de datos para representar un ficheiro cun tamaño de 1 Gbyte + 6 Mbytes + 40 Kbytes, diferenciando entre bloques de datos e bloques de índices.

$$\begin{aligned} \cdot \text{Tam. bloque} &= 4\text{Kbytes} \cdot 1024 = 4096 \text{ bytes} \\ \cdot \text{Tam. punteiro} &= 8 \text{ bytes} \end{aligned} \quad \left. \begin{array}{l} 4096/8 = 512 \text{ punteiros/bloque} \end{array} \right\}$$

$$\# \text{ bloques} = (10^6 + 6 \cdot 1024 + 40) / 4 = 263690 \text{ bloques}$$



- 2 Un proceso abre o ficheiro anterior (en P1) "/home/usr1/dir1/datos" (tamaño 1 Gbyte + 6 Mbytes + 40Kbytes) en modo lectura, cuxo número de hard links é 2. O usuario efectivo do proceso coincide co propietario do ficheiro e tamén ten os permisos de acceso os directorios home, usr1 e dir1. Os cachés de datos e inodos están inicialmente baleiros e a entrada usr1 está no séptimo bloque do directorio home, mentres que as outras entradas están no primeiro bloque dos seus directorios pai. Indique o seguinte sobre o anaco de código:

- (A) Cal é o valor asignado ao descriptor de ficheiro fd?:
- (B) Cal é o número de accesos ao disco necesarios, só na área de datos, ao abrir o ficheiro?:
- (C) Indica o número de bloques que o SO cómpre ler desde o disco para obter o valor de c en fgetc(fd):
- (D) Indica os permisos de ficheiro que se imprimen en formato rwxrwxrwx:
- (E) Cal é o número de ligazóns duros (hard links) de "datos" despois de executar unlink?:

```

struct stat buf;
char c;
chmod("/home/usr1/dir1/datos", 0442);
if (lstat ("/home/usr1/dir1/datos", &buf)!=-1){
    printf("%s", convertir_permisos(buf.st_mode)); /* se convierten los permisos a formato rwxrwxrwx y se imprimen*/
    /* se convierten los permisos a formato rwxrwxrwx e se imprimen*/
    /* permissions are converted to format rwxrwxrwx and printed*/
    int fd=open("/home/usr1/dir1/datos", O_RDONLY); /* es la primera apertura de fichero del proceso */
    /* ésta es la primera apertura de fichero do proceso */
    /* it is the first file open of the process*/
    lseek(fd, 1.073.741.824, SEEK_SET); /* (33.554.432=2^30) */
    /* SEEK_SET indica que el desplazamiento se considera a partir del origen del fichero */
    /* SEEK_SET indica que o desplazamiento considerase desde a orixe do ficheiro */
    /* SEEK_SET specifies that the offset is considered from the origin of the file */
    c=fgetc(fd);
    close(fd);
    unlink("/home/usr1/dir1/datos"); // Borrar o nome e o ficheiro
}

```

0	stdin
1	stdout
2	stderr
3	fd
4	

} por defecto

(A) O valor de fd é 3 (Pq 0, 1 e 2 están ocupados por defecto) →

(B) N° ACESOS AO DISCO = A.BLOQUE INODO + A.BLOQUE DATOS } Só nos preguntan para área de datos: 10



(C) 10 accesos directos } 10. 4 = 40KB } D (non chegamos a 1GB, SEGUIMOS)

Tam. bloques = 4 KB

TAM. TOTAL = 1GB

" 1.048.576 KB

512 · 4 = 2048 KB } IS (D+IS < 1GB, SEGUIMOS)

512 · 512 · 4 = 16B } ID (D+IS+ID > 1GB, PARAMOS)

} N° BLOQUES = 1 ID + 1 IS + DATA BLOCK = 3

(D) chmod("/home/usr1/dir1/datos", 0442);

PROPIETARIO	GRUPO	OUTROS
0 -----	-----	-----

rwx

442 → 100 100 010
r-- r-- -w-

Permiso	Valor Octal	Descripción
---	0	no se tiene ningún permiso
-x-	1	solo permiso de ejecución
-w-	2	solo permiso de escritura
-wx	3	permisos de escritura y ejecución
r--	4	solo permiso de lectura
r-x	5	permisos de lectura y ejecución
r-w-	6	permisos de lectura y escritura
rwx	7	todos los permisos establecidos, lectura, escritura y ejecución

(E) 1

3

Créanse 3 *hard link* a un ficheiro "f1" (cuxo número de inodo é 25634, tamaño 965 bytes e número de *hard links*=1) no mesmo directorio:

> ln f1 f2 // f1 = original f2 = hard link

> ln f1 f3

> ln f1 f4

Máis tarde, crease un *soft link* ao ficheiro f3:

> ln -s f3 slink /*crear soft link slink */

/* o comando ls -l amosaría slink -> f3 */

Cal é o tamaño (en bytes) do ficheiro "f3"? 965 bytes

Finalmente, se borra (comando rm) o ficheiro f2

> rm f2

Cal é agora o tamaño (en bytes) do ficheiro "slink"? 2 bytes, pq ao ser soft link non se copia o contido senón a ruta, que neste caso é f3 e ao ser 2 caracteres corresponde a 2 bytes de tamaño

4

Indique se é verdadeiro/falso en cada pregunta.

Cada apartado puntuá 0.075p. Cada resposta incorrecta puntuá -0.075p. Cuestiós non respondidas non puntuán. A puntuación mínima para P4 é 0, é dicir, en ningún caso P4 ten puntuación negativa para o exame total.

- A. Mientras un ficheiro está abierto, o S.O. guarda una copia del seu inodo na memoria principal. C
- B. O contido dun directorio (entradas de directorio) almacénase na área de datos. C
- C. O código *linkado* cunha biblioteca dinámica é "autocontido". F
- D. O código binario de *fork()* atópase na librería estándar de C (*libc*). F
- E. Es posible crear vínculos duros (*hard links*) entre diferentes sistemas de ficheiros montados. F
- F. O tipo de ficheiro "ligazón simbólica" (*symbolic link*) está codificado no campo "modo" do inodo (o mesmo campo no que tamén están codificados os permisos do ficheiro). C
- G. A idea fundamental dun sistema de ficheiros Unix basado en rexistro (*journaling file system*) es hacer un seguimiento de todos os ficheiros creados e eliminados. F
- H. A *Buffer Cache* minimiza los posibles errores, en caso de caída de enerxía, cuando se realizan operaciones de cambio de datos/metadatos no sistema de ficheiros. F

PARTE MEMORIA

1

VERDADEIRO ou FALSO

- Na pila do proceso está o código das funcións recursivas cuando se llaman
- Cuando se ejecuta un programa C, las variables locales de main() están en la pila del proceso
- Para un proceso varios hilos de ejecución (threads), los hilos comparten código, datos y pila.

- O código da función de librería malloc usado nun programa C está no espacio de direccións do proceso
- Para un sistema con táboa de páxinas en tres niveis (sin caché, nin TLB), traer e executar unha instrucción que engade un valor constante a un rexistro, implica exactamente tres accesos a memoria.
- Para un sistema con táboa de páxinas en tres niveis, donde a táboa de páxinas non está fixa en memoria, e procesador con caché e TLB, traer e executar unha instrucción que engade un valor constante a un rexistro, pode implicar catro accesos a memoria.
- Si as direccións físicas son de 32 bits e as páxinas de 4Kbytes, o número de páxina virtual necesariamente ven dado por 20 bits.
- En un sistema con segmentación que NON TEN paxinación (segmentación pura), é posible implementar memoria virtual
- Con Táboa de Páxinas Invertida, os sistemas operativos non podrían resolver os fallos de páxina dos procesos
- Para un proceso con un espacio virtual de N páxinas, o algoritmo de reemplazo FIFO produce sempre os mesmos fallos de páxina con N marcos asignados ó proceso que con $N+1$

2 Un proceso ten a cadea de referencias a páxinas que se mostra e ten asignadas tres marcos de memoria. As tres primeiras referencias, isto é, as referencias as páxinas E, D, H, producen necesariamente 3 fallos de páxina porque ningunha páxina do proceso estaba en memoria. ¿Cal é o número total de fallos de páxina que se producen con algoritmo de reemplazo LRU (Least Recently Used)? Obviamente hay que contar esos 3 fallos iniciais no total. Tanto a asignación de páxinas a frames como o total de número de fallos deben ser correctos para puntuar a pregunta.

E D H B D E D A E B E D E B G
 ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ } FALLOS PÁXINA = 9

Un proceso ten a cadea de referencias a páxinas que se mostra e ten asignadas tres marcos de memoria. As tres primeiras referencias, isto é, as referencias as páxinas E, D, H, producen necesariamente 3 fallos de páxina porque ningunha páxina do proceso estaba en memoria. ¿Cal é o número total de fallos de páxina que se producen con algoritmo de reemplazo LRU (Least Recently Used)? Obviamente hay que contar esos 3 fallos iniciais no total. Tanto a asignación de páxinas a frames como o total de número de fallos deben ser correctos para puntuar a pregunta.

E D H B D E D A E B E D E B D } 8 fallos de páxina

1) E E E B B B A A A A D D D *
 2) D D D * D D D D B B B B * B
 3) H H H E E E * E * E * E E

Marcos

3) Unha arquitectura de memoria ten direccions virtuais de 64 bits e direccions físicas de 52 bits. Hay un número de bits non usados nas direccions virtuais. Ten páxinas de 256 Kbytes (1 Kbytes = 1024 bytes), táboa de páxinas en dous niveis co mesmo número de bits nas direccions virtuais para cada un dos niveis e entradas na táboa de páxinas de 8 bytes. A) ¿Cántos bits NON usados hay nunha dirección virtual? B) ¿Cántos bits se usan nunha dirección virtual para seleccionar a entrada da táboa de páxinas raíz? Tanto os cálculos como o resultado final teñen que ser correctos para obter a puntuación

$$\# \text{ entradas en cada TP} = \frac{256 \cdot 1024}{8} = 2^{15} \text{ entradas/TP}$$

B) Empréganse 15 bits para seleccionar a entrada de 1P

64 bits - 48 bits = 16 bits libres

A) Dir. vir = 64 bits

$$\# \text{ entradas} \cdot \text{desprazamento} = \underbrace{2^{15}}_{N1} \cdot \underbrace{2^{15}}_{N2} \cdot \underbrace{2^{18}}_{\text{Tam. pax}} = 2^{48}$$

Unha arquitectura de memoria ten direccions virtuais de 64 bits e direccions físicas de 52 bits. Hay un número de bits non usados nas direccions virtuais. Ten páxinas de 128 Kbytes (1 Kbytes = 1024 bytes), táboa de páxinas en dous niveis con mesmo número de bits nas direccions virtuais para cada un dos niveis e entradas na táboa de páxinas de 8 bytes. A) ¿Cántos bits NON usados hay nunha dirección virtual? B) ¿Cántos bits se usan nunha dirección virtual para seleccionar a entrada da táboa de páxinas raíz? Tanto os cálculos como o resultado final teñen que ser correctos para obter a puntuación

$$\# \text{ entradas por TP} = \frac{128 \cdot 1024}{8} = 2^{14} \text{ entradas/páxina}$$

$$A) \text{ Tam. pax} = 128 \cdot 1024 = 2^{17} \longrightarrow 2^{14} \cdot 2^{14} \cdot 2^{17} = 2^{45} \longrightarrow 64 - 45 = 19 \text{ bits libres}$$

B) 14 bits de entrada

PARTE PROCESOS

1) VERDADEIRO ou FALSO

-La credencial real y la credencial efectiva de un proceso son siempre iguales entre si/Real credential always matches the effective credential/A credencial real e a credencial efectiva dun proceso sempre son iguais entre si F

-Las variables de entorno son iguales para todos los procesos del sistema/Environment variables are the same for all processes in the system/As variables de entorno son iguales para todos os procesos do sistema F

-Las variables de entorno son iguales para todos los procesos del mismo usuario/Environment variables are the same for all processes of the same user/As variables de entorno son iguales para todos os procesos do mesmo usuario F

-Cada proceso tiene su copia de los datos del kernel/Each process has its copy of the kernel data/Cada proceso ten a súa copia dos datos do núcleo F

-La pila del kernel debe estar protegida de accesos concurrentes/The kernel stack must be protected from concurrent accesses/A pila do núcleo debe estar protexida de accesos simultáneos **F**

-La transición de ejecución a espera es SIEMPRE desde ejecución en modo kernel/The transition from running to blocked is ALWAYS from kernel running/A transición de executar a esperar SEMPRE é dende executar en modo núcleo **V**

-El estado de apropiado (preempted) es el mismo que el de listo para ejecución/The preempted state is the same as runnable (ready to run) state/O estado apropiado (preempted) é o mesmo que listo para executar **V**

-Cuando un programa comienza su ejecución las variables de entorno están en la pila de usuario/When a program starts execution, environment variables are stored in the user stack/Cando un programa comeza a súa execución, as variables de entorno están na pila de usuario **V**

-La llamada execvp() crea un proceso que ejecuta un programa que está en el PATH/The execvp() system call creates a process that executes a program which is in the PATH
A chamada execvp() crea un proceso que executa un programa que está no PATH **F**

-Suponiendo que se llama desde el proceso padre del proceso pi2, waitpid(pi2,NULL,0) desasigna la estructura proc del proceso pi2 cuando termina/Assuming the calling process is pi2's parent process, waitpid(pi2,NULL,0) deallocates pi2's proc struct/Supoñendo que se chama desde o proceso pai do proceso pi2, waitpid (pi2, NULL, 0) deslocaliza a estrutura proc do proceso pi2 cando remata **V**

2

CONTESTA SE É CORRECTO OU NON

O seguinte código crea un proceso que executa un programa que se pasa cos seus parámetros nun array de punteiros rematado por NULL. (argv [0] é o nome do executable, argv [1] o primeiro parámetro ...)

```
void Execute (char *argv[])
{
    pid_t pid;

    if ((pid=fork())==0){
        if (execvp(argv[0],argv)==-1)
            perror (<<cannot execute>>);
    }
    else
        waitpid (pid,NULL,0);
}
```

INCORRECTO. En ningún momento se mata ao proceso fillo (falta un exit) polo que quedarán dous procesos executándose á vez.

```
void Execute (char *argv[])
{
    pid_t pid;

    if ((pid=fork())==0){
        if (execvp(argv[0],argv)==-1)
            perror (<<cannot execute>>);
        exit (0);
    }
    else
        waitpid (pid,NULL,WNOHANG);
}
```

CORRECTO. Neste caso a execución é en segundo plano (o pai non espera a que o fillo termine), xa que a flag de waitpid é WNOHANG, e non 0.

```

void Execute (char *argv[])
{
    pid_t pid;

    if (execvp(argv[0], argv)==-1){
        perror («cannot execute»);
        exit(0);
    }
    else
        waitpid (pid,NULL,0);
}

```

INCORRECTO Falta un fork, por lo que no se crea ningún proceso

3

En un sistema hay dos procesos A y B con duraciones 4-(5)-3 (ráfaga de CPU de 4, seguida de ráfaga de E/S de 5, seguida de ráfaga de CPU de 3) y 2-(4)-1, (ráfaga de CPU de 2, seguida de ráfaga de E/S de 4, seguida de ráfaga de CPU de 1) respectivamente. Los instantes de llegada son 0 para A y 1 para B.

Rellenar la siguiente tabla con el tiempo de retorno para A, tiempo de retorno para B y el porcentaje de uso de la CPU (en %), para los casos de multiprogramación con FCFS, multiprogramación con SRTF y no multiprogramación

	FCFS										SRTF					NO Multiprogram				
Tiempo Retorno A	12										14					12				
Tiempo Retorno B	12										7					18				
Uso de CPU (%)	$(13 - 3) / 13 \cdot 100 = 76,9\%$										71,4%					52,6%				

Como FCFS pone un solo proceso á vez en CPU e E/S



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
CPU	A	A	A	A	B	B				A	A	(A)	(B)							
E/S					A	A	A	A	B	B	B	B								

FCFS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
CPU	A	B	B	A	A	A	(B)			A	A	(A)	(A)							
E/S				A	A	B	B	B	B	A	A	A	A							

SRTF

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
CPU	A	A	A	A						A	A	(A)	B	B				(B)		
E/S				A	A	A	A	A	A					B	B	B	B			

NON MULTIP

• PARTE E/S

(OPCIÓN A)

1

A- El registro de control de un controlador de dispositivo permite saber si el dispositivo está disponible para recibir/transmitir un nuevo dato. F

O rexistro de control dun controlador de dispositivo permite saber se o dispositivo está dispoñible para recibir/transmitir un novo doto.
In a device controller, the control register permits us to know if the device is ready to receive/transmit a new piece of data.

B- Considérese la estructura en capas del software de E/S. Asumamos un disco duro con un sistema de ficheros que usa bloques de 1Kbyte. Cuando carguemos un fichero de 2500 bytes en un editor de textos, la capa denominada software independiente del dispositivo se encarga de calcular cuántos sectores de disco deben ser leídos: F

Considérese a estrutura en capas do software de E/S. Asumamos un disco duro cun sistema de ficheiros que usa bloques de 1Kbyte. Cando carguemos un ficheiro de 2500 bytes nun editor de textos, a capa denominada software independente de dispositivo encárgase de calcular cants sectores de disco deben ser lidos.

Let us consider the layered structure of the I/O software. Lets assume a hard drive containing a file system that uses 1Kbyte blocks. When we load a file of size 2500 bytes into a text editor, the layer named device-independent software is in charge of computing the number of sectors that must be read from disk.

C- Un disco duro con 4 caras y 16000 pistas por cara, tiene 64000 cilindros: F

Un disco duro con 4 caras e 16000 pistas por cara, ten 64000 cilindros.

A hard disk with 4 sides and 16000 tracks in each side contains 64000 cylinders.

D- Considerando la siguiente salida del ls: El major-number del dispositivo /dev/sdb es 8: V

Considerando a seguinte saída do ls: O major-number do dispositivo /dev/sdb é 8.

Considering the following output of the ls command: The major number of the device /dev/sdb is 8.

```
user@beowulf:~$ ls -l /dev/sd*
brw-rw---- 1 root disk 8, 0 nov 25 12:20 /dev/sda
brw-rw---- 1 root disk 8, 1 nov 25 12:20 /dev/sdal
brw-rw---- 1 root disk 8, 2 dic 31 01:25 /dev/sda2
brw-rw---- 1 root disk 8, 16 nov 25 12:20 /dev/sdb
brw-rw---- 1 root disk 8, 17 nov 25 12:20 /dev/sdb1
```

E- Un controlador DMA configurado en modo ráfaga tiene mayor prioridad de acceso al bus que otro configurado en modo robo de ciclos: V

Un controlador DMA configurado en modo ráfaga ten maior prioridade de acceso ao bus que outro configurado en modo roubo de ciclos.
A DMA controller configured in burst mode has a higher priority when accessing the bus than other one configured in cycle-stealing mode.

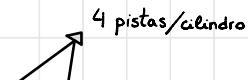
2

En un determinado instante, la cola de peticiones de E/S para acceder a cilindros de un disco contenía la peticiones: [25, 100, 360, 77, 125, 222]. Ya se han atendido las peticiones [100] (en primer lugar) y [77]. Indique cuál será el siguiente cilindro accedido si utilizamos los algoritmos:

SSTF: 125

C-SCAN: 25

SCAN: 25



3

Tenemos un disco que contiene 16384 (=16*1024) sectores. El disco tiene 2 platos y 2 caras en cada plato. Cada sector contiene 512 bytes, y cada pista contiene 16 sectores. ¿Cuántos cilindros tiene? ¿Cuántos bloques verá el S.O. si se formatea usando bloques de 2048 bytes?

$$\text{Nº cilindros} = 16384 \text{ sectores} \cdot \frac{1 \text{ pista}}{16 \text{ sectores}} \cdot \frac{1 \text{ cilindro}}{4 \text{ pistas}} = 256 \text{ cilindros}$$

$$2048 \text{ bytes} \cdot \frac{1 \text{ sector}}{512 \text{ bytes}} = 4 \text{ sectores/bloque} \longrightarrow 16384 \text{ sect} \cdot \frac{1 \text{ bloque}}{4 \text{ sect}} = 4096 \text{ bloques}$$

4

El fichero "fichero.txt" contiene "012345678901234567890\n". ¿Cuál será el contenido de BUF[i] (i=1..4) TRAS ejecutar la instrucción en la línea 08 del siguiente programa?

After/tras lin.02: BUF[i] =	-	-	-	-	\0
After/tras lin.08: BUF[i] =	0	1	2	3	\0
i =	0	1	2	3	4

```

1.01 int main(){
1.02     char BUF[5]= {'-', '-', '-', '-', '\0'};
1.03     int fd = open("fichero.txt",O_RDONLY);
1.04     int fd2 = dup(fd);
1.05     lseek(fd2, 2, SEEK_SET);
1.06     pread(fd,BUF,1,0);           0 - - - \0
1.07     lseek(fd, 2, SEEK_CUR);
1.08     1.08    pread(fd,BUF+1,3,1);   0 1 2 3 \0
1.09     printf("%s\n",BUF);
1.10     read(fd,BUF,3);           → Se non houbese este un comensáu
1.11     read(fd,BUF+2,2);          a len desde o pos 0, no que pread
1.12     printf("%s",BUF); close(fd); NON actualiza a posición
1.13 }
```

• PARTE E/S

(opción B)

1

A- El registro de datos de un controlador de dispositivo permite saber si el dispositivo está disponible para recibir/transmitir un nuevo dato: F

O rexistro de datos dun controlador de dispositivo permite saber se o dispositivo está disponível para recibir/transmitir un novo dato.
In a device controller, the data register permits us to know if the device is ready to receive/transmit a new piece of data.

B- Considérese la estructura en capas del software de E/S. La capa denominada software independiente de dispositivo es la encargada de chequear si un proceso tiene permisos para abrir un fichero: V

Considérese a estrutura en capas do software de E/S. A capa denominada software independente do dispositivo é a encargada de chequear se un proceso ten permiso para abrir un ficheiro.

Let us consider the layered structure of the I/O software. The layer named device independent software is in charge of checking if a process has enough permissions to open a file.

C- Un disco duro con 4 caras y 16000 cilindros, tiene 64000 pistas: V

Un disco duro con 4 caras e 16000 cilindros, ten 64000 pistas.

A hard disk with 4 sides and 16000 cylinders contains 64000 tracks.

D- Considerando la siguiente salida del ls: El major-number del dispositivo /dev/sda es 0: F

Considerando a seguinte saída do ls: O major-number do dispositivo /dev/sda é 0.

Considering the following output of the ls command: The major number of the device /dev/sda is 0.

```

user@beowulf:~$ ls -l /dev/sd*
brw-rw---- 1 root disk 8, 0 nov 25 12:20 /dev/sda
brw-rw---- 1 root disk 8, 1 nov 25 12:20 /dev/sdal
brw-rw---- 1 root disk 8, 2 dic 31 01:25 /dev/sda2
brw-rw---- 1 root disk 8, 16 nov 25 12:20 /dev/sdb
brw-rw---- 1 root disk 8, 17 nov 25 12:20 /dev/sdb1
```

E- Un controlador DMA configurado en modo bus transparente tiene mayor prioridad de acceso al bus que otro configurado en modo robo de ciclos: F

Un controlador DMA configurado en modo bus transparente ten maior prioridade de acceso ao bus que outro configurado en modo roubo de ciclos.

A DMA controller configured in transparent-bus mode has a higher priority when accessing the bus than other one configured in cycle-stealing mode.

- 2) En un determinado instante, la cola de peticiones de E/S para acceder a cilindros de un disco contenía las peticiones: [35, 100, 360, 77, 125, 222]. Ya se han atendido las peticiones [77] (en primer lugar) y [35]. Indique cuál será el siguiente cilindro accedido si utilizamos los algoritmos:

SSTF: 100

C-SCAN: 360

SCAN: 100

- 3) Tenemos un disco que contiene 32768 (=32*1024) sectores. El disco tiene 2 platos y 2 caras en cada plato. Cada sector contiene 512 bytes, y cada pista contiene 8 sectores. ¿Cuántos cilindros tiene? ¿Cuántos bloques verá el S.O. si se formatea usando bloques de 8192 bytes?

$$N^{\circ} \text{ pistas} = 2 \text{ platos} \cdot \frac{2 \text{ caras}}{1 \text{ plato}} = 4 \text{ pistas/cilindro}$$

$$N^{\circ} \text{ cilindros} = 32768 \text{ sectores} \cdot \frac{1 \text{ pista}}{8 \text{ sectores}} \cdot \frac{1 \text{ cilindro}}{4 \text{ pistas}} = 1024 \text{ cilindros}$$

$$8192/512 = 16 \text{ sectores/bloque} \longrightarrow 3278 \text{ sectores} \cdot \frac{1 \text{ bloque}}{16 \text{ sectores}} = 2048 \text{ bloques}$$

- 4) El fichero "fichero.txt" contiene "012345678901234567890\n". ¿Cuál será el contenido de BUF[i] (i=1..4) TRAS ejecutar la instrucción en la línea 08 del siguiente programa?

After/tras lin.02: BUF[i] =	-	-	-	-	\0
After/tras lin.08: BUF[i] =	4	5	0	-	\0
i =	0	1	2	3	4

<- Fill your answer here

<- Contesta aquí

```

1.01 int main()
1.02     char BUF[5]= {'-', '-', '-', '-', '\0'}
1.03     int fd = open("fichero.txt", O_RDONLY);
1.04     int fd2 = dup(fd);
1.05     lseek(fd2, 3, SEEK_SET);
1.06     pread(fd,BUF+2,1,0);           - - 0 - \0
1.07     lseek(fd, 1, SEEK_CUR);
1.08     pread(fd,BUF,2,4);           4 5 0 - \0
1.09     printf("%s\n",BUF);
1.10     read(fd,BUF,1);
1.11     read(fd,BUF+3,2);
1.12     printf("%s",BUF); close(fd);
1.13 }
```

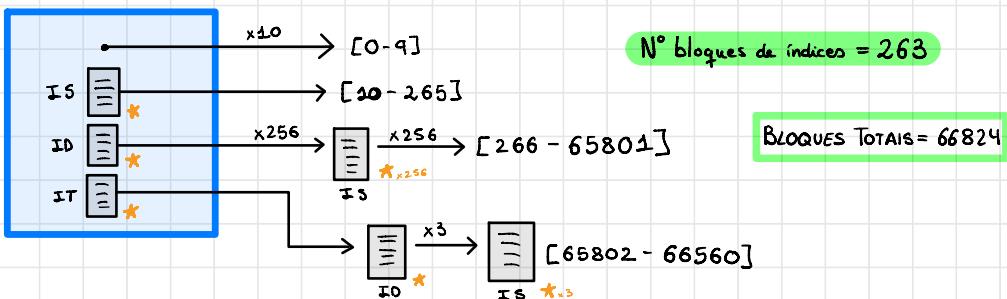
XANEIRO 2020

SISTEMAS DE FICHEIROS

- P1) Un sistema de archivos tipo UNIX System V tiene un tamaño de bloque de 2Kbytes, i-nodos con 10 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 8 bytes. Calcular cuántos bloques son necesarios en el área de datos para representar un fichero con un tamaño de 130 Mbytes+1byte, diferenciando entre bloques de datos y bloques de índices.

$$\begin{aligned} \text{Tam. bloque} &= 2 \cdot 1024 = 2048 \text{ bytes} \\ \text{Tam. punteiros} &= 8 \text{ bytes} \end{aligned} \quad \left. \begin{array}{l} \frac{2048}{8} = 256 \text{ punteiros/bloque} \end{array} \right\}$$

$$N^{\circ} \text{ bloques} = (130 \cdot 1024^2 + 1) / 2048 = 66561 \text{ bloques de datos}$$



- P2) En ese sistema de archivos UNIX (tamaño de bloque de 2Kbytes), el *boot* ocupa los 3 primeros bloques de su partición de disco. Por tanto, el *superbloque* comienza en el bloque lógico 3 de la partición de disco del SF (se numera a partir del bloque 0). El fichero "datos" tiene asociado el *inode* 643 y ocupa el bloque (lógico) 33 desde el principio de la partición. El tamaño del *inode* es de 64 bytes. Calcula el tamaño (en Kbytes, no bloques) del *superbloque* y el *bloque lógico* (desde el principio de la partición) correspondiente al *inode* 3201.

boot	superbloq	lista inodes	Data area
0 1 2 3 ... ?			→ 13

$$\text{Inodos/bloque} = \frac{2 \cdot 1024}{64} = 32 \text{ inodos/bloque}$$

$$634 / 32 \approx 20 \text{ bloques ate a } 33 \longrightarrow 33 - 20 = 13 \text{ } \overset{?}{\boxed{\text{l\'astimo bloque de superbloque}}} \rightarrow \text{TAM} = 10 \cdot 2 = 20 \text{ Kb}$$

$$3201 / 32 \approx \underbrace{100 \text{ bloques}}_{\text{Da lista de inodos}} \longrightarrow 100 + 13 = \boxed{\text{bloque 113}}$$

- P3) En el sistema de archivos del problema P1 (tamaño bloque 2Kbytes, 10 punteros directos, ...), tenemos el archivo “`/home/juan/so/p1/p1.c`” con un tamaño de 3Mbytes e inodo número 6549. Al ejecutar un proceso (con el código principal indicado a continuación), el usuario efectivo del proceso coincide con el propietario del fichero (`p1.c`), y tiene además los permisos de acceso y lectura a los directorios `home`, `juan`, `so` y `p1`. Las cachés de datos e inodos están inicialmente vacías y la entrada `p1` está en el octavo bloque de su directorio padre (`so`), mientras los demás entradas están en el primer bloque de su directorio padre. Indicar lo siguiente referente al trozo de código:

```
struct stat buf, char c;  
chmod("/home/juan/so/p1/p1.c", 0420);  
if (lstat ("/home/juan/so/p1/p1.c", &buf)!=-1){  
    printf("%s", convertir_permisos(buf.st_mode)); /* se convierten los permisos a  
formato rwxrwxrwx y se imprimen*/  
    int fd1=open("/home/juan/so/p1/p1.c", O_RDONLY); /* es la primera apertura  
de fichero del proceso */  
    link("p1.c", "practica1.c"); /* se crea hard link practica1.c */  
    symlink("practica1.c", "slink_practica1.c"); /* se crea link simbólico a practica1.c */  
    int fd2=open("/home/juan/so/p1/practica1.c", O_RDONLY); /* segunda apertura */  
    lseek(fd2, 2097152, SEEK_SET); /* SEEK_SET indica que el desplazamiento  
se considera a partir del origen del fichero (2097152 = 221) */  
    c=fgetc(fd2);  
    close(fd2); close(fd1);  
    unlink("p1.c"); }
```

- A. ¿Cuál es el número de accesos necesarios a disco, únicamente en el área de datos, en la primera apertura del fichero `p1.c`? **12**
- B. ¿Cuál es el valor asignado al descriptor de fichero `fd2`? **4**
- C. Indica el número de bloques que el S.O. necesita leer en disco para obtener el valor de `c` en `fgetc(fd2)`: **11 bytes** (indicar unidad: bytes, Kbytes, ...)
- D. ¿Cuál es el tamaño del fichero “`slink_practica1.c`”? **11 bytes**
- E. `fgetc` es una llamada al sistema operativo. Cierto/Falso: **F**
- F. `link` es una llamada al sistema operativo. Cierto/Falso: **C**
- G. `unlink` va a eliminar la entrada de directorio “`p1.c`” en su directorio padre pero no libera el inodo 6549. Cierto/Falso: **C**
- H. El lincador encuentra el código de `printf` en la librería estándar de C (versión estática o dinámica de la librería). Cierto/Falso: **C**
- I. Indica los permisos del fichero `p1.c` impresos en formato `rwxrwxrwx`: **r - - - -w - - -**

$$\text{C) } \text{TAM} = \boxed{2048 \text{ KB}}$$

$$256 \cdot 2 = \boxed{512 \text{ KB}} \text{ en IS}$$

$$\text{A. Directos} = 12 \quad \boxed{24 \text{ KB}} \text{ en D}$$

$$256 \cdot 256 \cdot 2 = \boxed{131072 \text{ KB}} \text{ en ID (D+IS+ID > TAM ✓)}$$

$$\begin{array}{l} \text{Tam. bloque} = 2 \text{ KB} \\ \text{Tam. punteros} = 8 \text{ bytes} \end{array} \quad \boxed{256 \text{ punteros / bloque}}$$

Nº de bloques = 3

• MEMORIA

- 1 -En la pila del proceso están las variables locales y parámetros de funciones: _____
- Las variables locales del main no están en la pila del proceso: _____
- Las variables automáticas definidas en funciones están en la pila del proceso: _____
- Las variables automáticas definidas en main() no están en la pila del proceso: _____
- El código y los datos de la librería malloc usada en un programa C están en el espacio de direcciones del kernel: _____
- Las variables estáticas definidas en funciones están en la pila del proceso: _____
- Las variables estáticas definidas en main() no están en la pila del proceso: _____
- Para un proceso con varios hilos de ejecución, cada hilo tiene su propio espacio de direcciones (código, datos, stack): _____
- Para un sistema con una tabla de páginas en un nivel (y no caché, no TLB), traer y ejecutar una instrucción que añade un valor constante a un registro, implica exactamente dos accesos a memoria:

- Para un sistema con una tabla de páginas de dos niveles (y no caché, no TLB), traer y ejecutar una instrucción que añade un valor constante a un registro, implica exactamente dos accesos a memoria:

- Para un sistema con tablas de páginas en dos niveles donde la página raíz no está fija en memoria y procesador con caché y TLB, traer y ejecutar una instrucción que añade un valor constante a un registro, puede implicar ningún acceso a memoria: _____
- Dado un número de bits para las direcciones virtuales, y una tabla de páginas de un nivel, el tamaño de la tabla de páginas se reduce con páginas más grandes: _____
- Si las direcciones físicas son de 32 bits y las páginas de 4Kbytes, el número de página física viene dado por 18 bits: _____ F _____
- Una tabla de páginas multinivel típicamente reduce la cantidad de memoria necesaria para almacenar tablas de páginas comparada con una tabla de páginas de un nivel: _____ V _____
- Si el bit de validez es 1 a cero en una Entrada de Tabla de Página necesaria para un acceso a memoria, la página deseada deberá ser traída a memoria desde el dispositivo de almacenamiento: _____ F _____
- Con Tablas de Páginas Invertida, las páginas virtuales pueden ser más grandes que las páginas físicas: _____ F _____
- Las TLBs son más beneficiosas con tablas de páginas multinivel que con tablas de páginas de un nivel: _____ V _____
- Cuando el .ty bit está a cero en una Entrada de Tabla de Página necesaria para un acceso a memoria, existe una copia exacta de la página deseada en el dispositivo de almacenamiento: _____ V _____

-El algoritmo de reemplazo LRU con N+1 páginas de memoria siempre se comporta mejor que LRU con N páginas de memoria: _____

-El algoritmo de reemplazo FIFO con N+1 páginas de memoria siempre se comporta mejor que FIFO con N páginas de memoria: _____

X

2. a) (0.50 puntos) Considera un sistema con un direccionamiento de memoria física de 8GBytes, tamaño de página de 8Kbytes y tamaño de una entrada de tabla de páginas de 4 bytes. ¿Cuántos niveles de tablas de páginas son necesarios para manejar direcciones virtuales de 46 bits si cada tabla páginas es de tamaño una página? Debe indicar los cálculos para que la respuesta puntue. Número de niveles: 3

→ b) (0.25 puntos) Para la solución anterior, ¿de cuantos bits de control (bit referencia, bit R/W, dirty bit, etc) y bits no usados (se pide la suma total de ambos tipos), se dispone en una entrada de tabla de página de cada nivel? Debe indicar los cálculos para que la respuesta puntue.

→ c) (0.25 puntos) Imagine que con la solución anterior el procesador dispone de caché de datos e instrucciones y TLB, en este caso, para una operación de lectura de un byte de memoria (debe dar la explicación correcta para que la respuesta puntue).

¿cuál sería el número mínimo de accesos a memoria?: 0

¿cuál sería el número máximo de accesos a memoria?: 4

$$PTE = 32 = 4 \text{ bytes} \cdot 8 \text{ bits}$$

a) N° entradas = $\frac{8 \cdot 10^24}{4} = 2048 = 2^{11} \text{ entradas / TP}$

$$1 \text{ TP} \rightarrow 2^1 \cdot 2^{13} = 2^{24} \text{ bytes}$$

$$2 \text{ TP} \rightarrow 2^2 \cdot 2^{13} = 2^{35} \text{ bytes}$$

$$3 \text{ TP} \rightarrow 2^3 \cdot 2^{13} = 2^{46} \text{ bytes} \rightarrow \text{Falta 3 niveles para tener direcciones virtuales de 46 bits}$$

• PROCESOS

① FACER NO BOLETÍN

2. Un sistema monoprocesador con múltiples colas tiene tres colas: SY para procesos del sistema, IT para procesos de usuario interactivos y NI para procesos de usuario no interactivos. La planificación entre las colas es por prioridades apropiativas (un proceso de una cola solo podrá ejecutarse si no hay procesos listos en las colas de mayor prioridad) siendo la prioridad más alta para la cola de procesos del sistema y la más baja para los procesos de usuario no interactivos. En el cuadro siguiente se muestran los procesos del sistema en un instante dado ($x-(y)-z$ representa una ráfaga de CPU de duración x, seguida una de e/s de duración y seguida de otra de CPU de duración z). Suponemos que el sistema usa Round Robin de quantum 3 para los procesos del sistema, Round Robin de quantum 1 para los interactivos y que los no interactivos se planifican con SJF

	Ráfagas		Tiempo de llegada		Tipo	
Proceso A	4	-7	-2	0		sistema
Proceso B	2	-7	-4	1		sistema
Proceso C	2	-3	-2	(3)-1	0	interactivo
Proceso D	1	-3	-1	(3)-1	2	interactivo
Proceso E		4		0		no interactivo
Proceso F		3		2		no interactivo

Rellenar el siguiente cuadro con la planificación

CPU	A	A	A	B	B	A	C	D	C	F	F	D	B	B	B	B	A	A	(A)	(B)	C	(D)	C	(F)	E	E	C	E	(E)
SY		B	B	4	A									A	A	B													
IT	C	C	C	c	c	c	D	D	D	D				C	C	C	C	D	D	D	D	C	C	D	D	C			
NI	E	E	E	c	c	c	F	P	F	F	F	F	E	E	E	E	F	F	F	F	F	E	E	F	F	E	E	E	
E/S							B	B	B	B	B	B	A	A	A	A	D	D	D	D					c	c	c		

• ENTRADA/SAÍDA

1. (0.50 puntos) Disponemos de un disco duro con 2000 cilindros numerados del 0 al 1999. La cabeza está situada en el cilindro 605 atendiendo una petición de acceso al cilindro 605 justo después de haber atendido la petición anterior en el cilindro 602.

Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: (400, 300, 1550, 900, 201, 495), indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

SSTF:	495	400	300	201	900	1550
SCAN:	900	1550	495	400	300	201
C-LOOK:	900	1550	201	300	400	495

3. (0.25 puntos) Disponemos de un disco con un plato y dos caras por plato que contiene 2048 cilindros, y 512 sectores por pista. Cada sector contiene 512bytes.

¿cuántos bloques ve el sistema operativo si lo formateamos con bloques de 2048 bytes?

Ponga una "X" en el recuadro correspondiente y justifique su respuesta.

$$\rightarrow 2048 / 512 = 4 \text{ sectores/bloque}$$

 1024; 2048; 4096;

Otro (indique abajo cuál)

Justifique brevemente su respuesta: *Cada cilindro tiene cuatro pistas.*

$$2048 \cdot \frac{2}{1} \cdot \frac{512}{1} \cdot \frac{1}{4} = 2^{19} \text{ bloques}$$

4. (0.25 puntos) El fichero "fichero.txt" contiene: "012345678901234546789\n", Indique cuál será el contenido de BUF[i] ($i \in \{0..4\}$) TRAS ejecutar la intrucción en la línea "09" del siguiente programa:

Tras 1.09: BUF[i]=	3	4	-	-	\0
posición i=	0	1	2	3	4

```
1.01. #include "includes.h"
1.02. int main(){
1.03.     char BUF[5] = {'-', '-', '-', '-', '\0');
1.04.     int fd = open("fichero.txt", O_RDONLY);
1.05.     int fd2 = dup(fd);
1.06.     lseek(fd2, 2, SEEK_SET);
1.07.     pread(fd, BUF, 1, 0);           //leo "0", BUFF contiene 0,-,-,-,\0 (lseek no afecta a pread)
1.08.     lseek(fd, 2, SEEK_CUR);
1.09.     pread(fd, BUF, 2, 3);        //leo "34", BUFF contiene 3,4,-,-,\0
1.10.     read(fd, BUF, 1);
1.11.     read(fd, BUF+3, 2);
1.12.     printf("%s", BUF); close(fd);
1.13. } //Nota: sabemos que al ejecutar dicho código no se produjo ningún error de ejecución.
```

XANEIRO 2019

FICHEIROS

P1) Un sistema de archivos tipo *system V* tiene un tamaño de bloque de 4 Kbytes e inodos con 10 direcciones directas de bloques, una indirecta simple, una indirecta doble y una indirecta triple. Además, utiliza direcciones de bloques de 4 bytes. Consideremos el fichero "p1.c" en el directorio /home/usr1/so/p1, con un tamaño de 4Gbytes+6 Mytes+45 KBytes.

Calcular cuántos bloques de disco son necesarios para representar ese archivo. Indicar también cuánta fragmentación interna (en Kbytes) se produce.

$$\# \text{ blogues} = \underbrace{(4 \cdot 1024^3 + 6 \cdot 1024^2 + 45 \cdot 1024)}_{\text{Tam jich}} / \underbrace{(4 \cdot 1024)}_{\text{Tam blog}} = 1050124 \text{ blogues}$$

$$\text{Fragmentación interna} = \underbrace{4 \cdot 2024}_{\text{Tam. bloque}} - (\text{Tam. fich mod tam. bloq}) = 3 \text{kbytes}$$

P2) Supongamos la siguiente secuencia de comandos desde el directorio /home/usr1/so/p1:

- i) Se crea un *hard link* al fichero *p1.c* (cuyo número de inodo es 11545, tamaño 4Gbytes+6 Mytes+45 KBytes y num. de *hard links*=3):
In p1.c practica1.c / crea hard link practica1.c */*
 - ii) Posteriormente se crea un *soft link* al archivo *practica1.c*:
In -s practica1.c slink / el comando ls -l mostraría slink -> practica1.c */*
 - iii) Finalmente se crea un *hard link* al fichero *slink*:
In slink blinkslink / crea hard link blinkslink */*

Contestar a lo siguiente:

- A. Indicar cuál es el número de (*hard*) *links* del fichero *hlinkslink* después de las tres operaciones: **λ**
 - B. Indicar el tamaño del fichero *hlinkslink*: **11**
 - C. Una vez creados los ficheros *slink* y *hlinkslink*, al borrar el fichero *p1.c* (*rm p1.c*) se puede seguir accediendo al contenido del fichero con número de inodo 11545 a través del *link simbólico* *slink* (Cierto/Falso): **C**
 - D. Con un sistema de ficheros Unix basado en registro (*journaling file system*), al borrar el fichero *p1.c*, primero se realiza una copia de su inodo en el registro antes de liberar sus bloques del área de datos. (Cierto/Falso): **F**

P3) Después de crear el *hard link* *practica1.c* en el apartado i) del ejercicio previo, un proceso abre ese archivo "/home/usr1/so/p1/practica1.c" en modo lectura 2 veces seguidas. El usuario efectivo del proceso coincide con el propietario del fichero (*practica1.c*), y tiene además los permisos de acceso y lectura a los directorios *home*, *usr1*, *so* y *p1*. Las cachés de datos e inodos están inicialmente vacías y la **entrada p1** está en el **sexta bloque** de su directorio padre (*so*), mientras **las demás** entradas están en el **primer bloque** de su directorio padre. Indicar lo siguiente referente al trozo de código:

```
struct stat buf;
char c;
chmod("/home/usr1/so/p1/practica1.c", 0664);
if (lstat ("/home/usr1/so/p1/practica1.c", &buf)!=-1){
    printf("%s", convertir_permisos(buf.st_mode)); /* se convierten los permisos a
    formato rwxrwxrwx y se imprimen*/
    int fd1=open("/home/usr1/so/p1/p1.c", O_RDONLY); /* es la primera apertura
    de fichero del proceso */
    int fd2=open("/home/usr1/so/p1/practica1.c", O_RDONLY); /* segunda apertura */
    lseek(fd2, 524288, SEEK_SET); /* SEEK_SET indica que el desplazamiento
    se considera a partir del origen del fichero
    (524288 = 219) */
    c=fgetc(fd2);
    close(fd2); close(fd1);
}
```

- A. ¿Cuál es el número de accesos necesarios a disco, únicamente en el área de datos, en la apertura del fichero *practica1.c*? **10**
- B. ¿Cuál es el valor asignado al descriptor de fichero *fd2*? **4**
- C. Indica el número de bloques que el S.O. necesita leer en disco para obtener el valor de *c* en *fgetc(fd2)*:
- D. *fgetc* es una llamada al sistema operativo. (Cierto/Falso): **F**
- E. Indica los permisos del fichero *p1.c* impresos en formato *rwxrwxrwx*: **rw-rw-r--**

(C) Tam. bloque = 4KB }
 Tam. puntero = 4 bytes } **1024 punteros / bloque**

TAM = 512KB

1024 · 4 = 4096KB en IS (D+IS>TAM ✓)

A. Directos = 10 → 10 · 4 = **40KB** en D

Nº BLOQUES = 2

P4) En ese sistema de archivos UNIX (tamaño de bloque de 4Kbytes), el tamaño de la lista de inodos en disco es de 16 Mbytes. El superbloque mantiene un mapa de bits de inodos libres para determinar los inodos libres/ocupados de la lista de inodos. Ese mapa de bits, que es parte del superbloque, **ocupa un total de 8 bloques**. Calcular el tamaño (en bytes) de un inodo.

8 bits para mirar que
inodos están libres/ocupados

$$\frac{16 \cdot 1024^2}{8 \cdot 4 \cdot 1024 \cdot 8} = 64 \text{ bytes}$$

→ Pasar a bits

Tam. lista inodos	(bytes)
Tam. mapa bits	(bits)

MEMORIA

2. Considera el siguiente código y de la respuesta correcta (0.2 puntos)

```
#include <csdlib.h>
#include <stdio.h>
#include <string.h>
char *f(char *a) {
    static char *b;
    b=a;
    return b;
}
int main () {
    char af[30] = "Hello fun";
    printf("%s");
    strcpy(a, "Bye fun");
    printf("%s");
}
```

- a) Da error en tiempo de ejecución por asignar una variable global (a) a una estática (b)
- b) Da error en tiempo de ejecución por asignar una variable local (a) a una estática (b)
- c) Producen la salida pretendida, esto es:
Hello f
Bye f
porque nada lo impide
- d) Produce la salida
Hello f
Hello f
porque b es estática y no puede cambiar de valor
- e) Da error en tiempo de compilación porque printf no puede llevar de argumento una función
- f)Ninguna de las anteriores es cierta

Respuesta: C

4. (0.6 puntos) Supón un esquema de paginación con dos niveles de tablas de páginas, con páginas de 4Kbytes, entradas en las tablas de páginas de 4 bytes y direcciones virtuales de 32 bits. Si no hay información suficiente para responder alguna pregunta, indíquelo.

¿Cuál es el formato de las direcciones virtuales?

$$\text{Tam. páginas} = 4\text{Kbytes} = 2^{12} \text{ bytes} \longrightarrow \boxed{12 \text{bits de posición}}$$

$$\text{Dir. VIR} = 32 \text{ bits} = \boxed{12 + 10 + 10}$$

offset TPA TPZ

¿Cuantos bits de control y reservados hay en cada entrada de una tabla de páginas?

En cada entrada de 4bytes se usa un número de bits para indicar el **número de frame** y el **resto para bits de control y reservados** pero se desconoce esta información

-¿Cuál sería el tamaño máximo en tablas de páginas que puede ocupar un proceso?

$$1025 = 1 + 2^{10}, \text{ es decir, } 1025 \text{ páginas de 4Kb, o } (2^{12} \text{ bytes} + 2^{22} \text{ bytes}), \text{ o } 4\text{Kbytes} + 4\text{Mbytes}$$

-Para un proceso con 3 páginas (una de código, una de datos, una de pila) y con la disposición habitual del espacio de direcciones virtuales, ¿cuál es el tamaño ocupado en tablas de páginas para este proceso?

La TP de primer nivel. La primera entrada de la TP de primer nivel apunta a una TP de segundo nivel que apunta a las páginas de código y datos. La última entrada de la TP de primer nivel apunta a una TP de segundo nivel que apunta a la página de pila. Por tanto 3 páginas en tablas de páginas, es decir 12Kbytes.

-Para el proceso del apartado anterior, si la página de datos no está en la memoria física, ¿cuál es el tamaño ocupado en tablas de páginas para este proceso?

Idem que la cuestión anterior

-¿Cuál es el tamaño del espacio de direcciones físicas expresado en número de frames?

Se desconoce el número de bits reservado para indicar el número de frame, por tanto no se puede responder a esta cuestión

3. Considera el siguiente código y de la respuesta correcta (0.2 puntos)

```
#include <csdlib.h>
#include <stdio.h>
#include <string.h>
char *f(char *a) {
    static char *b;
    b=a;
    return b;
}
int main () {
    char *a = malloc(sizeof(char)*30);
    strcpy(a,"Hello fun");
    printf("%s");
    strcpy(a, "Bye fun");
    printf("%s");
}
```

- a) Da error en tiempo de ejecución por asignar una variable dinámica (a) a una estática (b)
- b) Da error en tiempo de ejecución por asignar una variable local (a) a una estática (b)
- c) Da error en tiempo de compilación porque printf no puede llevar de argumento una función
- d) Producen la salida pretendida, esto es:

Hello f

Bye f

porque nada lo impide

e) Produce la salida

Hello f

Hello f

porque b es estática y no puede cambiar de valor

f)Ninguna de las anteriores es cierta

Respuesta: D

5. (0.3 puntos) En un sistema con memoria virtual y segmentación paginada, suponga que la programación es correcta, es decir, que las invocaciones de free llevan como argumento un puntero obtenido con malloc y que nunca se ha accedido a bytes de memoria que no fuesen obtenidos con malloc. Una buena práctica de programación es liberar con free todos los bloques de memoria asignados con malloc antes de que un proceso termine. Para estas invocaciones de free de la respuesta correcta, no es necesario justificarla

- a) algún free puede provocar fallo de segmento por acceso página inválida
- b) algún free puede provocar fallo de segmento por acceso a segmento inválido
- c) **algun free puede provocar fallo de página por acceso a una página no usada en mucho tiempo**
- d) algún free puede provocar fallo de segmento si la página fue modificada y salvada al dispositivo de swapping
- e) algún free puede provocar fallo de página si la página fue modificada y no salvada al dispositivo de swapping
- f) ninguna de las anteriores es cierta

PROCESOS

Se hai POLO MENOS UN fork, sempre vai haber ÚNICAMENTE UN pid = 0.

1. Considérese el siguiente código en C (se supone que contiene todos los ficheros include necesarios y que compila correctamente)

```
main(int argc, char * argv[])
{
    int i=100;

    pid=fork()+fork(); //*****
    if (pid==0)
        execl("./a.out", "./a.out",NULL);

    printf ("i vale: %d\n",i);
}
```

a) *i vale : 100
i vale : 100
i vale : 100
i vale : 1*

b) *A misma*

donde el código de ./a.out es el siguiente: (también se supone contiene todos los ficheros include necesarios y que compila correctamente)

```
int i;
main (int argc, char *argv[])
{
    i++;
    printf ("i vale: %d\n",i);
}
```

- a) *¿Qué salida produce dicho código?* (suponemos que tanto fork() como exec() se completan correctamente)
- b) *¿Qué salida produciría si sustituimos las llamadas fork() por llamadas ifork() en la línea marcada con //*****/?*

1. Considérese el siguiente código en C (se supone que contiene todos los ficheros include necesarios y que compila correctamente)

```
main(int argc, char * argv[])
{
    int i=100;

    pid=fork()*fork(); //*****
    if (pid>0)
        execl("./a.out", "./a.out",NULL);

    printf ("i vale: %d\n",i);
}
```

a) *el valor de i es: 1*

*i vale : 100
i vale : 100
i vale : 100
i vale : 100*

Nova pid=0 e outra pid<0

b) *i vale: 100
i vale: 100*

Non hai ningún fork, é o pai

donde el código de ./a.out es el siguiente: (también se supone contiene todos los ficheros include necesarios y que compila correctamente)

```
int i;
main (int argc, char *argv[])
{
    i++;
    printf ("el valor de i es: %d\n",i);
}
```

- a) *¿Qué salida produce dicho código?* (suponemos que tanto fork() como exec() se completan correctamente)

- b) *¿Qué salida produciría si una de las llamadas fork() produce un error (exec() se completa correctamente)?*

- c) *¿Qué salida produciría si las dos llamadas fork() producen un error (exec() se completa correctamente)?*

2. Un sistema tiene 4 procesos, A, B, C, D cuyas duraciones son 1-(6)-3, 2-(6)-1, 3-(5)-6 y 1-(1)-4 respectivamente (x-(y)-z indica una ráfaga de CPU de duración x, seguida de una ráfaga de e/s de duración y, seguida de una ráfaga de CPU de duración z). El sistema tienen dos colas de planificación: la cola de alta prioridad que se planifica con Round Robbing

de quanto 2 y la cola de baja prioridad que se planifica por SJF. Un proceso de la cola de baja prioridad solo se ejecuta cuando no hay lista ningun proceso de la cola de alta prioridad, suponemos además que todos los procesos llegan en el instante 0 en el orden A,B,C,D y que pueden realizar e/s concurrentemente. Rellenar el siguiente cuadro con la planificación, indicando el cada instante que proceso está en CPU, listo en la cola de Alta Prioridad (HPQ), listo en la cola de Baja Prioridad(LPQ) o en E/S. Suponemos que **A** y **B** son procesos de Alta Prioridad y que **C** y **D** son procesos de Baja Prioridad

CPU	A	B	B	D	C	C	C	A	A	B	A	D	D	D	D	C	C	C	C	C
HPQ	B										A									
LPQ	C	C	C	C				D	D	D	D	D	D	D		C	C	C		
E/S		A	A	A	A	A	A	B	B	B	B	C	B	C	c	c	c			

Calcular además los tiempos de retorno y espera para cada proceso.

	Ráfagas	Tiempo Retorno	Tiempo de Espera
Proceso A	1-(6)-3	11	1
Proceso B	2-(6)-1	10	1
Proceso C	3-(5)-6	21	7
Proceso D	1-(1)-4	15	9

→ Tempo en colas

3. ¿Cual o cuales, si es que alguna, de las siguientes afirmaciones son ciertas en un sistema unix, referidas a la pila del kernel?

- (a) La batería que mantiene la configuración del kernel cuando el sistema se apaga
- (b) Zona de memoria usada para para pasar parámetros a funciones y variables locales cuando un proceso se ejecuta en modo kernel Definición
- (c) Lo que proporciona un pulso de corriente al procesador para cambiar a modo kernel
- (d) Zona de memoria donde se almacenan las variables de entorno y las credenciales de los usuarios del sistema
- (e) Solo hay una en el sistema
- (f) Hay una para cada procesador/núcleo
- (g) Hay una para cada proceso
- (h) Debe ser de solo lectura
- (i) Hay una para cada thread Se o proceso ten varios threads e o SO permite que varios threads dun mesmo processo realicen chamadas ao sistema concurrentemente

ENTRADA / SAÍDA

1. (1.00 puntos) Disponemos de un disco duro con 1000 cilindros numerados del 0 al 999. Sabemos que en las dos últimas peticiones atendidas se accedió a los cilindros 402 y al 400 respectivamente, y que la cabeza está ahora situada sobre el cilindro 400.

Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: (501, 625, 250, 552, 99, 800), indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

FCFS o FIFO:	501	625	250	552	99	800
SSTF:	501	552	625	800	250	99
SCAN:	250	99	501	552	625	800
C-SCAN:	250	99	800	625	552	501
C-LOOK:	250	99	800	625	552	501

2. (0.5 puntos) El fichero "fichero.txt" contiene: "012345678901234546789\n", y sabemos que al ejecutar el siguiente código no se produjo ningún error de ejecución:

```
#include "includes.h"
int main(){
int fd = open("fichero.txt",O_RDONLY);
lseek(fd, 5, SEEK_SET);
char BUF[5] = {'-', '-', '-', '-', '\0'};

#ifdef CASO_A
read(fd,BUF,1);
read(fd,BUF+2,2);

5 - 67

#endif
#ifdef CASO_B
pread(fd,BUF,1,0);
lseek(fd, 2, SEEK_CUR);
pread(fd,BUF,2,3);
read(fd,BUF,1);

Ø 34 → 74 --

#endif
printf("%s",BUF); close(fd);
}
```

567 → 89

Asumiendo que el contenido del buffer **BUF** ($BUF[i], i = 0 \dots 4$) en función de si está definido CASO_A, CASO_B, o CASO_C es el siguiente:

CASO_A:	CASO_B:	CASO_C:																															
<table border="1"> <tr><td>5</td><td>-</td><td>6</td><td>7</td><td>\0</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> </table>	5	-	6	7	\0	0	1	2	3	4	<table border="1"> <tr><td>7</td><td>4</td><td>-</td><td>-</td><td>\0</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> </table>	7	4	-	-	\0	0	1	2	3	4	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> </table>	0	1	2	3	4	0	1	2	3	4	→ 89 -- \0
5	-	6	7	\0																													
0	1	2	3	4																													
7	4	-	-	\0																													
0	1	2	3	4																													
0	1	2	3	4																													
0	1	2	3	4																													
BUF[i], i=	i=	i=																															

Indíquese en qué casos, el estado de **BUF** representa la realidad: Indique V/F (Verdadero/Falso) y razoné los casos en los que considere que es Falso.

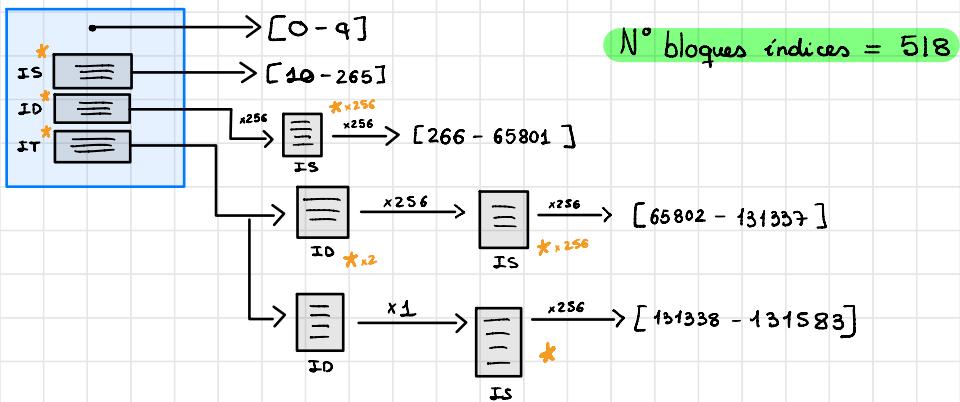
XANEIRO 2018

P1) Un sistema de archivos tipo system V tiene un tamaño de bloque de 2 Kbytes e inodos con 10 direcciones directas de bloques, una indirecta simple, una indirecta doble y una indirecta triple. Además, utiliza direcciones de bloques de 8 bytes. Consideremos el fichero "datos" en el directorio /home/usr1/dir1, con un tamaño de 257 MBytes.

Calcular cuántos bloques de disco son necesarios para representar ese archivo de 257 MBytes.

$$2 \cdot 1024 / 8 = 256 \text{ punteiros / bloque}$$

$$\# \text{ bloques} = 275 \cdot 1024^2 / 2 \cdot 1024 = 131584 \text{ bloques de datos}$$



P2) Supongamos la siguiente secuencia de comandos desde el directorio home/usr1/dir1:

- Se crea un **soft link** al archivo **datos** (cuyo **número de inodo es 45689**, **tamaño 257 MBytes** y num. de **hard links=1**, con el comando **In -s datos slink** /*el comando ls -l mostraría slink -> datos */)
- Posteriormente se crea un hard link al fichero **slink**:
In slink hlink /* crea hard link **hlink** */
- Una vez creados los ficheros **slink** y **hlink**, se borra el fichero **datos** (**rm datos**).

Contestar a lo siguiente:

- A. El inodo de *slink* es el 45689 (Cierto/Falso): **Falso** } 0 inodo cambia
- B. El tamaño de *hlink* es 14 bytes (Cierto/Falso): **Falso** } & 5 bytes
- C. Tras el borrado del fichero datos (*rm datos*) se libera su inodo (queda como libre en la lista de inodos en disco) (Cierto/Falso): **Cierto**
- D. Con un sistema de ficheros Unix basado en registro (*journaling file system*), al borrar un fichero, parte de la información del inodo se guarda en el registro. (Cierto/Falso): **Falso**

P3) Un proceso abre el archivo anterior “*/home/usr1/dir1/datos*” en modo lectura, cuyo número de **hard links** es **1**. El usuario efectivo del proceso coincide con el propietario del fichero, y tiene además los permisos de acceso al directorio *home*, *usr1* y *dir1*. Las cachés de datos e inodos están inicialmente vacías y la entrada *home* está en el **tercer bloque** del directorio raíz, mientras **los demás** entradas están en el **primer bloque** de su directorio padre. Indicar lo siguiente referente al trozo de código:

```
struct stat buf;
char c;
chmod("/home/usr1/dir1/datos", 0640);
if (lstat ("/home/usr1/dir1/datos", &buf)!=-1){
    printf("%s", convertir_permisos(buf.st_mode)); /* se convierten los permisos a formato
                                                 rwxrwxrwx y se imprimen*/
    int fd=open("/home/usr1/dir1/datos", O_RDONLY); /* es la primera apertura de fichero del
                                                 proceso */
    lseek(fd, 67108864, SEEK_SET); /* SEEK_SET indica que el desplazamiento
                                                 se considera a partir del origen del fichero
                                                 (67108864 = 226) */
    c=fgetc(fd);
    close(fd);
}
```

- A. ¿Cuál es el valor asignado al descriptor de fichero *fd*? **3**
- B. ¿Cuál es el número de accesos necesarios a disco, únicamente en el área de datos, en la apertura del fichero?: **6**
- C. Indica el número de bloques que el S.O. necesita leer en disco para obtener el valor de *c* en *fgetc(fd)*: **3**
- D. *fgetc* es una función de librería de C. Cierto/Falso: **Verdadero**
- E. Indica los permisos del fichero impresos en formato *rwxrwxrwx*: **rw-r-----**

(C) Tam = 65536 Kb
256 punteiros/bloque
Tam. bloque = 4 Kb

D → 10 · 4 = **40kb**
IS → 256 · 4 = **1024 kb**
ID → 256 · 1024 = **262144kb** ✓
Tres accesos a bloques

P4) Calcular qué **número de bloque lógico** corresponde al **inodo del raíz** (se comienza a contar en el bloque lógico 0), y cuál bloque lógico **al inodo del fichero datos**, suponiendo lo siguiente:

- i) El nº de inodo del “/” es el 2, y al fichero **datos** le corresponde el inodo nº 202 (los inodos se comienzan a numerar a partir de 1).
- ii) El tamaño de un inodo es de 64 bytes (tamaño bloque = 2Kbytes).
- iii) El boot ocupa 2 bloques y el superbloque 9 bloques.



$$N^{\circ} \text{ inodos/bloque} = \frac{2 \cdot 1024}{64} = 32 \text{ inodos/bloque}$$

"/" está no bloque 11 (pq $2 < 32$)

"datos" está no bloque 17 (pq $\frac{202}{\frac{inodo}{inodos/bloque}} \approx 7$)

MEMORIA

1. (0.5 puntos) Considere el siguiente código C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char** f();
char** g;

int main(void)
{
    char a[]="Hello function f";
    char **b=NULL;
    b = f(a);
    g = f(a);
    printf("Cadena[%s] Dirección[%p] \n", a, &a);
    printf("Cadena[%s] Dirección[%p] \n", *b, b);
    printf("Cadena[%s] Dirección[%p] \n", *g, g);
}

return 0;
}

char** f(char *a)
{
    static char *c;
    c = (char *) malloc(strlen(a)+1);
    strcpy(c, a);
    return &c;
}
```

→ Está en PILA pq é uma variável local

→ É uma variável global pq está declarada fora do main

Una vez compilado y enlazado, la ejecución de este código imprime 3 veces la cadena "Hello function f" por pantalla y tres direcciones en hexadecimal (directiva %p). Considere las zonas de direcciones de la memoria de un proceso: P Pila, H Heap, y la zona de variables globales y estáticas en la que se distinguen dos zonas según están inicializadas o no, es decir, GSI Variables globales y estáticas inicializadas, GSN Variables globales y estáticas no inicializadas (llamada históricamente BSS). Las tres direcciones que se imprimen por pantalla se corresponden con las siguientes zonas:

- a) La primera de P, la segunda y la tercera de H.
- b) La primera de GSI, la segunda y la tercera de H.
- c) La primera de P, la segunda y la tercera de GSN.
- d) La primera de GSI, la segunda y la tercera de GSN.
- e) Las tres de P.
- f) Ninguna de las anteriores es cierta.

2. (0.5 puntos) Considere la siguiente cadena de referencia a páginas de un proceso:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 4

¿Cuántos fallos de página se producen si el proceso tiene asignadas 6 frames (páginas físicas), contando todos los fallos incluidos los que se producen al principio de la cadena que no implican reemplazo, y considerando que las frames inicialmente están vacías?. Rellena también la tabla con la asignación de páginas a frames. Tanto el número de fallos como la asignación de páginas a frames deben ser correctas para puntuar el apartado.

LRU:

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	4
1	1	1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
2	2	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	5	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	4
	6	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
F	F	F	F		F	F				F					F		F		

8 fallos

FIFO:

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	4
1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	5	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	4
	6	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
F	F	F	F		F	F				F					F		F		

11 fallos

Óptimo:

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	4
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	*	-	-	-	-	-	-	-
	6	-	-	-	-	-	-	-	-	-	-	*	-	-	-	-	-	-	-
F	F	F	F		F	F				F					F		F		

7 fallos

→ Pense no oco do 5
pq nunca más se
vuelve a usar

3. a) (0.3 puntos) Considera un sistema con 32Gbytes de memoria física, páginas de 8Kbytes y entradas en las tablas de páginas de 8 bytes. Con un nivel la TP ocupa una página. Con varios niveles la TP raíz o la de un nivel que no es el último, apunta a páginas que contienen TPs. ¿Cuál es el tamaño en bytes del espacio virtual direccionable si se usa tablas de páginas de 1, 2 o 3 niveles?

b) (0.2 puntos) ¿Cuál es el número mínimo de bits necesarios en cada entrada de una tabla de páginas para poder direccionar toda la memoria física?

De la respuesta y a continuación los cálculos

Número de bits necesarios: _____

c) (0.2 puntos) ¿Cómo es tamaño y formato de las direcciones lógicas en la arquitectura de 3 niveles?

a) TP 1 nivel:

$$\left. \begin{array}{l} \text{Tam. nivel TP} = 2^{13} \text{ bytes} \\ \text{Tam. entradas} = 2^3 \text{ bytes} \end{array} \right\} \boxed{\text{Nº entradas} = 2^{10}} \rightarrow \begin{array}{l} \text{Cada entrada apunta} \\ \text{a una página con } 2^{13} \\ \text{bytes} \end{array}$$
$$\underbrace{\text{Nº entradas} \cdot \text{Tam. pax}}_{2^{10} \cdot 2^{13} = 2^{23} \text{ bytes}}$$

TP 2 niveis:

$$\text{Nº entradas} = 2^{20} \longrightarrow \text{Tam. espacio direccionable} = 2^{20} \cdot 2^{13} = 2^{33} \text{ bytes}$$

TP 3 niveis:

$$\text{Nº entradas} = 2^{30} \longrightarrow 2^{43} \text{ bytes}$$

b)

$$\left. \begin{array}{l} \text{Mem. física} = 2^{35} \text{ bits} \\ \text{Tam. pax. físicas} = 2^{13} \text{ bits} \end{array} \right\} \boxed{2^{35}/2^{13} = 2^{22} \text{ páginas}} \quad \boxed{22 \text{ bits de direccionamiento}}$$

c) Son de 43 bits. 10 bits para cada nivel da TP (30) e os 13 restantes para o offset.

- X** 4. (0.3) Considere una arquitectura de tres niveles de TP como la del ejercicio anterior, un SO con memoria virtual y la situación en la que un proceso está en la cola de procesos listos para ejecución.

- a) Las páginas de código de ese proceso tienen que estar en memoria.
- b) Las páginas de código y pila de ese proceso tienen que estar en memoria.
- c) Las páginas de código, datos y pila de ese proceso tienen que estar en memoria.
- d) No es necesario que ninguna página de código, datos o pila de ese proceso esté en memoria.
- e) Un registro del procesador dedicado a dar acceso a la TP, debe apuntar a la dirección virtual de la tabla de páginas de primer nivel.
- f) Un registro del procesador dedicado a dar acceso a la TP, debe apuntar a la dirección física de la tabla de páginas de primer nivel.
- g) Ninguna de las anteriores es cierta.

Indique la respuesta correcta y razonelo. **Tanto la respuesta como el razonamiento deben ser correctos para puntuar la pregunta.**

Respuesta correcta: _____ d _____

Imagine que el proceso se acaba de crear y que memoria virtual es con paginación por demanda. Cuando el proceso tome la CPU y se ejecute la primera instrucción se producirá un fallo de página que traerá la primera página del proceso a memoria y sucesivos fallos de páginas irán cargando en memoria las páginas necesarias. En este momento f) sería cierto pero no en la situación del enunciado, en esa situación habrá otro proceso en CPU y ese registro contiene dirección física de la TP de primer nivel de ese proceso.

PROCESOS

1. Considérese el siguiente código en C (se supone que contiene todos los ficheros include necesarios y que compila correctamente)

```
main(int argc, char * argv[])
{
    int i=100;
    pid_t pid;

    pid=fork(); /******/
    execl("./a.out", "./a.out",NULL); } O par e o filho o executan
    printf ("i vale: %d\n",i); } Ninguén o executa
}
```

donde el código de ./a.out es el siguiente: (también se supone contiene todos los ficheros include necesarios y que compila correctamente)

```
int i;
main (int argc, char *argv[])
{
    i++;
    printf ("i vale: %d\n",i);
}
```

- a) ¿Qué salida produce dicho código?

i vale: 1
i vale: 1

- b) ¿Qué salida produciría si sustituimos la llamada *fork()* por una llamada *vfork()* en la línea marcada con /******/?

i vale: 1
i vale: 1

vfork diferenciase de *fork* en que se suspende ao pai ata que o filho fai unha chamada a *_exit()* ou *execve()*

Explicación: La llamada *fork()* crea un proceso, por tanto hay DOS procesos que llegan a la llamada *execl*. *execl* REEMPLAZA el espacio de direcciones por el de ./a.out. En ./a.out i es una variable externa sin inicialización explícita, y por tanto inicializada a 0, se incrementa y se imprime (el *printf* posterior a *exec* NO SE EJECUTA). Si sustituimos el *fork()* por *vfork()* la salida es la misma pues *vfork()* es una optimización de *fork()* en la que el proceso padre presta su espacio de direcciones al proceso hijo HASTA que se hace *exec*

2. Un sistema tiene 4 procesos, A, B, C, D cuyas duraciones son 3-(4)-1, 1-(6)-2, 4-(5)-3 y 6-(1)-1 respectivamente (x-(y)-z indica una ráfaga de CPU de duración x, seguida de una ráfaga de e/s de duración y, seguida de una ráfaga de CPU de duración z). El sistema tienen dos colas de planificación: la **cola de alta prioridad** que se planifica con **Round Robbing** de **cuanto 2** y la **cola de baja prioridad** que se planifica por **FCFS**. Un proceso de la cola de baja prioridad solo se ejecuta cuando no hay listo ningún proceso de la cola de alta prioridad, suponemos además que todos los procesos llegan en el instante 0 en el orden A,B,C,D y que pueden realizar e/s concurrentemente. Rellenar el siguiente cuadro con la planificación, indicando el cada instante que proceso está en CPU, listo en la cola de Alta Prioridad (HPQ), listo en la cola de Baja Prioridad(LPQ) o en E/S. Suponemos que **A y B** son procesos de **Alta Prioridad** y que **C y D** son procesos de **Baja Prioridad**

CPU	A	A	B	A	C	C	C	C	A	B	B	D	D	D	D	D	D	C	C	C	D
HPQ	B		A																		
RR(2)																					
LPQ	C	C	C	C	D	D	D	D	D	D	D	C	C	C	C	D	D				
FCFS	D	D	D	D																	
E/S					B	B	B	B	B	C	C	C	C	C	C			D			

Calcular además los tiempos de retorno y espera para cada proceso.

	Ráfagas	Tiempo Retorno	Tiempo de Espera
Proceso A	3-(4)-1	9	1
Proceso B	1-(6)-2	11	1
Proceso C	4-(5)-3	20	8
Proceso D	6-(1)-1	21	13

3. Un sistema en tiempo real está atendiendo 4 eventos periódicos de **periodos**

$T_1 = 100ms$, $T_2 = 50ms$ y $T_3 = 200ms$ $T_4 = 10ms$ y cuyos tiempos de

procesamiento son respectivamente $C_1 = 5ms$, $C_2 = 2ms$ $C_3 = 100ms$ y

$C_4 = 3ms$. Se quiere que atienda a un quinto evento periódico que ocurre

cada 300ms. ¿Cuál es la duración máxima de procesamiento de este evento

para que el sistema siga siendo planificable?

$$\sum_i \frac{C_i}{T_i} \leq 1$$

ENTRADA / SAÍDA

1. (0.75 puntos) Complete las líneas 1.9 y 1.10 para que la salida del comando `ls` en la línea 1.14 se almacene en el fichero `out.txt`. Complete también las líneas 1.20 y 1.21 para que la salida del comando `ls` en la línea 1.25 se muestre por la salida estándar.

```
1. 0 #include "includes.h"
1. 1 int main() {
1. 2     char * command[] = {"./bin/ls", "-l", "/", NULL};
1. 3     int fd, backup, status; pid_t pid;
1. 4
1. 5     unlink("out.txt");
1. 6     if ( (fd=open("out.txt", O_CREAT|O_EXCL|O_WRONLY,0777) )== -1) {
1. 7         perror("OPEN FAILED:");
1. 8         exit(0);
1. 9         backup= dup (STDOUT_FILENO); //respuesta
1.10         close (STDOUT_FILENO); //respuesta
1.11
1.12     dup(fd);
1.13     if (fork() ==0) {
1.14         if (-1 == execv("./bin/ls", comand)){
1.15             perror("command failed:");
1.16             exit(0);
1.17         }
1.18         waitpid(pid,&status);
1.19
1.20         close (STDOUT_FILENO); //respuesta
1.21         dup (backup); //respuesta
1.22
1.23         close(fd);close(backup);
1.24         if (fork() ==0) {
1.25             if (-1 == execv("./bin/ls", comand)){
1.26                 perror("command failed:");
1.27                 exit(0);
1.28             }
1.29 }
```

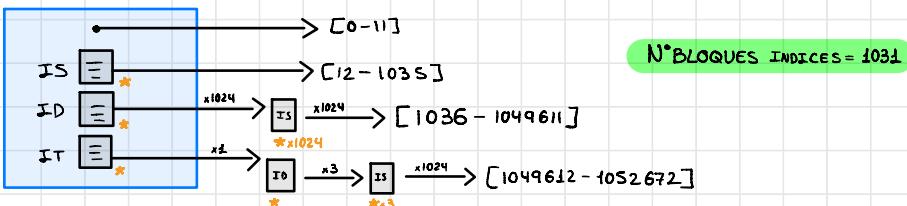
2. (0.75 puntos) En un determinado instante la cola de peticiones de E/S para acceder a cilindros de un disco duro contenía las peticiones: `(33, 341, 367, 440, 278, 101)`. Desconocemos el cilindro X que estaba siendo accedido justo antes de estas peticiones, pero sabemos que X no estaba entre los cilindros de la lista de peticiones. Si sabemos que el orden de atención de dichas peticiones fue: `(367, 440, 33, 101, 278, 341)`. Indíquese qué algoritmo/s de planificación de accesos a disco pudo/pudieron haber sido utilizados. Razone brevemente su respuesta. Adicionalmente, indíquese también UN posible número de cilindro (X) que podría estar siendo accedido antes de atender a dichas peticiones.

algoritmo	posible cilindro X	breve justificación de la respuesta
C-SCAN ou C-LOOK	$X \in [342, 366]$	<i>Se vai nun sentido (ascendente) e como non sabemos cuales son os extremos poden ser tanto C-SCAN como C-LOOK por igual.</i>

XULLO 2021

FICHEIROS

- P1) Un sistema de archivos tipo UNIX System V tiene un tamaño de bloque de 8Kbytes, i-nodos con 12 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 8 bytes. Calcular cuántos bloques son necesarios en el área de datos para representar un fichero con un tamaño de 8 Gbytes + 32 Mbytes + 1024 bytes, diferenciando entre bloques de datos y bloques de índices.



$$8 \cdot 1024 / 8 = 1024 \text{ punteros / bloque}$$

$$\text{nº bloques} = 105\,267\,3$$

- P2) Un proceso abre 2 veces el archivo anterior (en P1) "/home/usr1/datos" (tamaño 8 Gbytes + 32 Mbytes + 1024 bytes) en modo lectura, cuyo número de hard links es 2. El usuario efectivo del proceso coincide con el propietario del fichero, y tiene además los permisos de acceso a los directorios home y usr1. Las cachés de datos e inodos están inicialmente vacías y la entrada usr1 está en el séptimo bloque del directorio home, mientras las demás entradas están en el primer bloque de sus directorios padre. Indicar lo siguiente referente al trozo de código:

```
struct stat buf; char c1, c2;
chmod("/home/usr1/datos", 0752);
printf("%s", convertir_permisos(0752));
/* se convierten los permisos en octal a formato rwxrwxrwx y se imprimen */
/* se convierten los permisos en octal a formato rwxrwxrwx e se imprimen */
/* permissions in octal are converted to format rwxrwxrwx and printed*/
link ("/home/usr1/dir1/datos", "/home/usr1 /datos2")
int fd1=open("/home/usr1 /datos", O_RDONLY);
/* es la primera apertura de fichero del proceso */
/* it is the first file open of the process */
int fd2=open("/home/usr1 /datos2", O_RDONLY);
iseek(fd2, 4.000.000, SEEK_SET); /*SEEK_SET indica que el desplazamiento se considera a partir del origen del fichero */
/* SEEK_SET indica que o desplazamiento considerase desde a origen do ficheiro */
/* SEEK_SET specifies that the offset is considered from the origin of the file */
c1=fgetc(fd1); c2=fgetc(fd2);
close(fd1); close(fd2);
unlink('/home/usr1 /datos');
```

Cada apartado puntuúa 0.1p/ Each question scores 0.1p.

- A. ¿Cuál es el valor asignado al descriptor de fichero fd2?: 4
Cal é o valor asignado ao descriptor de fichero fd2:
What is the value assigned to the file descriptor fd2?
- B. ¿Cuál es el número de accesos necesarios a disco, únicamente en el área de datos, en la primera apertura del fichero?: 1
Cal é o número de accesos ao disco necesarios, só na área de datos, ao abrir o ficheiro a primeira vez?:
What is the number of disk accesses required, only in the data area, on the first file opening?:
- C. Indica el número de bloques que el SO necesita leer en disco para obtener el valor de c2 en fgetc(fd2):
Indica o número de bloques que o SO cómpre ler desde o disco para obter o valor de c2 en fgetc (fd2):
Indicate the number of blocks that the OS requires to read from disk to get the value of c2 in fgetc (fd2):
- D. Indica los permisos del fichero que se imprimen en formato rwxrwxrwx: 752 → r-w-r-x-w-
Indica os permisos de ficheiro que se imprimen en formato rwxrwxrwx:
Indicate the file permissions that are printed in rwxrwxrwx format:
- E. ¿Cuál es el número de hard links de "datos" después de ejecutar unlink?: 2
Cal é o número de ligazóns duros (hard links) de "datos" despois de executar unlink?:
What is the number of hard links of "data" after running unlink?

C) TAM = 4000000 bytes

D →

P3) Supongamos la siguiente secuencia de comandos:

1. Se crea un soft link al fichero "practica.c" (cuyo número de inodo es 22342, tamaño 455 bytes y num. de hard links=3) en su mismo directorio:

`ln -s practica.c slink` /* el comando `ls -l` mostraría `slink -> practica.c` */

2. Posteriormente se crea un hard link al fichero `slink`:

`ln slink hard_slink` /* crea entrada `hard_slink` */

Contestar a lo siguiente:

A. Indicar el tamaño (bytes) del fichero `hard_slink`: **10**

B. Indicar cuál es el número de (hard) links del fichero `hard_slink`: **2**

C. Una vez creados los ficheros `slink` y `hard_slink`, al borrar el fichero `practica.c` (`rm practica.c`) se decrementa su número de hard links. ¿Se puede acceder al contenido del fichero con número de inodo 22342 a través del link simbólico `slink`? (Sí/No): **No**

D. Misma pregunta a través del fichero `hard_slink`: (Sí/No) **No**

MEMORIA

2.v1. Un proceso tiene la cadena de referencias a páginas que se muestra y tiene asignados tres marcos de memoria. Las cuatro primeras referencias, estos es, la referencias a las páginas 4, 3, 4, 5, producen necesariamente 3 fallos de página porque ninguna página del proceso estaba en memoria. ¿Cuál es el número total de fallos de páginas que se producen con el algoritmo de reemplazo FIFO Segunda Oportunidad? Obviamente hay que contar esos 3 fallos iniciales en el total. Tanto la asignación de páginas a frames como el total de número de fallos deben ser correctos para puntuar la pregunta.

4	3	4	5	1	4	2	1	3	4	1	4
4*	4*	4*	4*	1*	1*	1*	1*	3*	3*	3*	3*
	3*	3*	3*	3	4*	4*	4*	4	4*	4*	4*
			5*	5	5	2*	2*	2	2	1*	1*
F	F		F	F	F	F		F		F	

