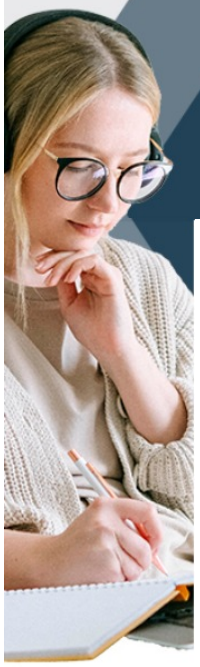
Apuntes SO.pdf

JDTadasGOT

Sistemas Operativos

3º Grado en Ingeniería Informática

Facultad de Informática
Universidad Complutense de Madrid

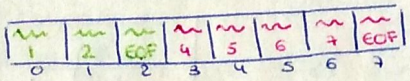


Almacenamiento en disco.

FAT

Mapa de bits: indica los bloques ocupados.

Tabla fat: indica donde están los bloques.



Se suele usar en pen-drives.

Mbits: 

La primera vez funciona muy bien porque los accesos se almacenan en orden, pero cuando se empieza a cambiar el tamaño de los ficheros se empieza a realizar la fragmentación

I-NODOS

Mapa de bits: igual que en FAT.

Tabla de bloques: donde busco información sobre i-nodos.

Tabla de i-nodos: donde busco información sobre bloques.

Tabla de bloques

BLQ 2	
.	3
..	5
UAR	7

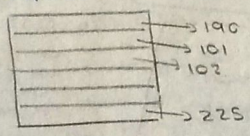
Soy el i-nodo 3
mi padre es el 5
y UAR está en
el i-nodo 7

Tabla de i-nodos

Nº INODO → 7
TIPO ← Fichero / Directorio → D
Nº ENLACES ≥ 1 (solo si es fichero) → NA
DIRECTO → 9 (bloque)
DIRECTO2
INDIRECTO
INDIRECTO2

Bloque indirecto

Contiene direcciones a bloques directos. Usado cuando se necesita más espacio.



Si se necesita aún más espacio, se usan bloques indirectos dobles, que contienen direcciones a bloques de direcciones a bloques directos

Febrero 14

1) 4096 bytes. Punteros de 32 bits. 8 directos, 1 indirecto y 1 indirecto²

Tabla de i-nodos:

	A1	A2	A3	A5	A6	a8	a9	a10	a11
nodo-i	1	2	3	4	6	7	9	10	12
enlaces	NA	NA	NA	1	NA	NA	1	2	1
tipo	D	D	D	F	D	D	F	F	F
D	4	5	6	13	8	9	14	12	15

Bloques:

4

.	1
..	1
A1	2
A2	3
a3	4

5

.	2
..	1
A5	6

6

.	3
..	1
A6	7

8

.	6
..	2
a8	9
a9	10

9

.	7
..	3
a10	10
a11	12

¿Tamaño máximo del fichero? Árbol.

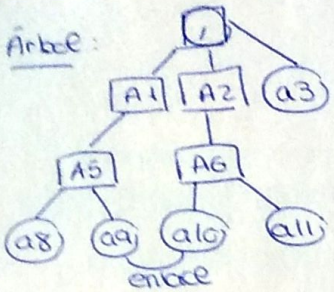
4096 → 4KB → $2^{10} \cdot 2^2 = 2^{12}$ bytes

32 bits → $32/8 = 4 = 2^2$ bytes

$\frac{2^{12}}{2^2} = 2^{10} = 1024$ punteros

$4KB \times (8 + 1024 + 1024^2) \approx \boxed{4\,GB}$

Árbol:



Si fueran 64 bits → 64/8 = 8 = 2^3

$\frac{2^{12}}{2^3} = 2^9 = 512$ punteros

$4KB(8 + 512 + 512^2) = \boxed{1\,GB}$

PREGUNTA: Si todo el disco lleno de ficheros de 7KB y bloques de 2KB → ¿espacio desaprovechado?

2KB	~
2KB	~
2KB	~
2KB	

1/8

Si bloques de 3KB

3KB	
3KB	
3KB	
3KB	

1/4

Enlaces

Enlace rígido

Tienen el mismo i-nodo (como a10 y a9 en el ejemplo). No se sabe cual es el original.

Enlace simbólico

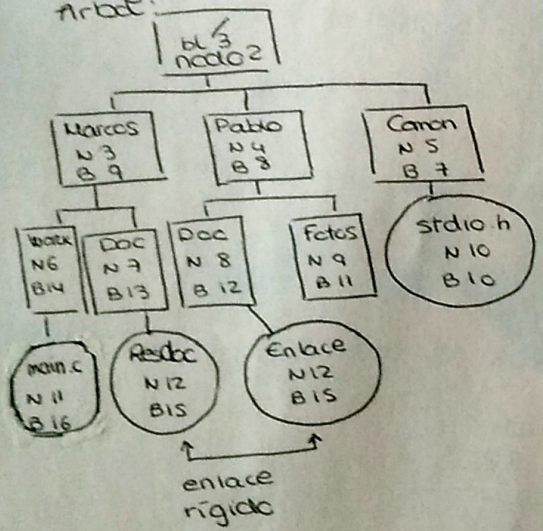
Parece el original pero en realidad contiene la ruta de otro. Tiene su propio i-nodo. Es sólo de lectura, es muy débil.

Comando: ln (-s) origen destino.
↳ simbólico.

Febrero 15

2000 bytes, punteros de 32 bits. D 10 I 1 I^2 1. Bloques 2KB

Árbol:



/
nodo 2
tipo D
enlaces NA
D 3

BLQ 3	
.	2
..	2
Marcos	3
Pablo	4
Canon	5

4
nodo 4
tipo D
enlaces NA
D 8

BLQ 8	
.	4
..	2
Doc	8
Fotos	9

8
nodo 8
tipo D
enlaces NA
D 12

BLQ 12	
.	8
..	4
Enlace	12

12
tipo F
enlaces 2
D 15

BLQ 15

a) Tam max (igual que el Feb 14)

b) Lleno de ficheros de 7KB

2KB	
2KB	
2KB	
2KB	

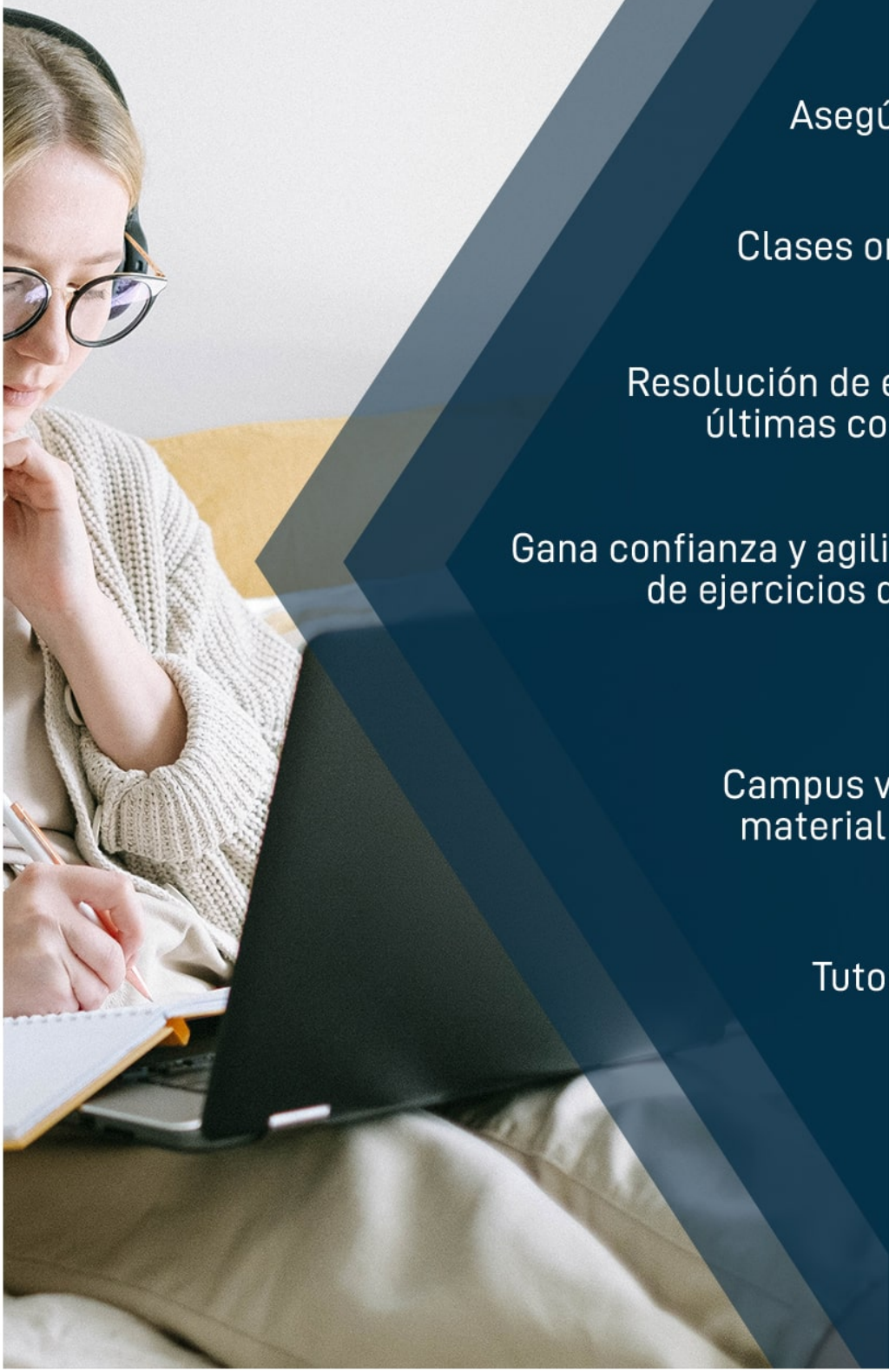
1/8

Maths informática

Academia especializada en estudios de la Facultad de Informática

CURSOS INTENSIVOS PARA EXÁMENES DE

CONVOCATORIA ORDINARIA Y EXTRAORDINARIA



Asegúrate un aprobado

Clases online en directo

Resolución de exámenes de las últimas convocatorias

Gana confianza y agilidad en la resolución de ejercicios de exámenes

Campus virtual con material de ayuda

Tutorías y resolución de dudas

www.mathsinformatica.com

Febrero 2016

1) 16 bytes bloques. Punteros de 8 bits.

Mapa de bits

1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 ... 0

Tabla de i-nodos

nodo	2	3	4	5	6	7	8	9	10
		opt	tmp	VAR	LoL	OKOV			
enlaces	NA	NA	NA	NA	~~1~~ 2	1			
tipo	D	D	D	D	F	F			
D1	3	5	6	0	1	10			
D2					2				
I					(4) → 9				

Bloques relevantes

opt

BLOQUE 2

.	2
..	2
opt	3
tmp	4

BLOQUE 5

.	3
..	2
LoL	6

tmp

BLOQUE 6

.	4
..	2
var	5
itanab	6
humbold	6

BLOQUE 4

.	9

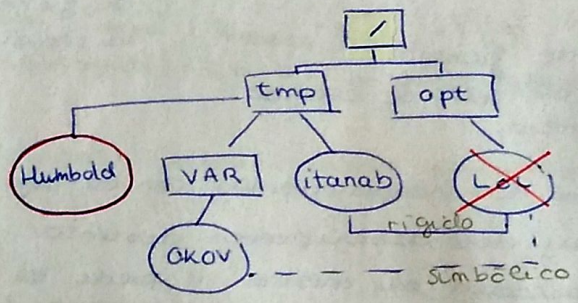
1) cd /
2) mkdir tmp/VAR
3) echo "And the rest is rust and stardust" > /opt/LoL (35 bytes)
4) ln /opt/LoL /tmp/itanab
5) ln -s /opt/LoL /tmp/VAR/OKOV

Posible estado final y árbol.

VAR

.	5
..	4
OKOV	7

1) No hace nada
2) var en el bloque 5, le ponemos un i-nodo y un bloque libre del mapa de bits
3) Bloque 6, me queda sin bloques directos. Caben 32, necesito 35. Usar bloque indirecto
4) No creo inodo, lo añado al bloque 6 y aumenta número de enlaces.
5) Si creo inodo 7. Cabe en un solo bloque.



→ mv /opt/LoL /tmp/Humbold El enlace simbólico no funciona.

Ficheros

- open (nombre, opciones, permisos) → opciones → O_RDWR ... → permisos
  - O_CREAT
  - O_TRUNC

  U G R
  LWX LWX LWX
  110 110 110 → 0666

- write (descriptor, buffer, bytes necesarios)
  No rebobina el puntero.

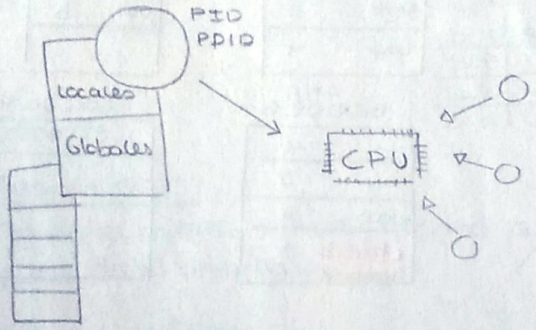
- lseek (. . .) → SEEK_SET : principio + nº bytes
  → SEEK_CUR : donde estoy + nº bytes
  ↳ se usa para saber donde estoy (con suma 0)

Procesos

Programa en ejecución.

Identificador de proceso: PID

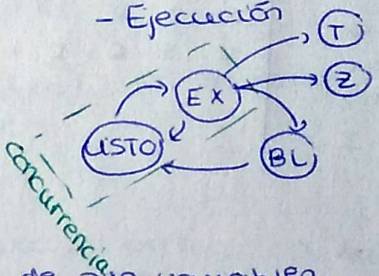
Identificador de proceso del padre: PPID



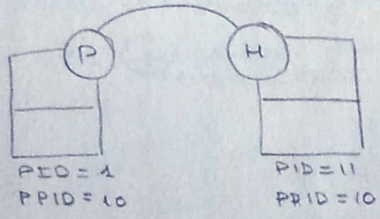
Un montón de procesos quieren ejecutarse en la CPU.

Posibles estados:

- Listo
- Bloqueado
- Ejecución



- fork() → clon de un proceso en el estado actual de sus variables.



Se establece una relación de padre - hijo.

No comparten absolutamente nada entre ellos, salvo los descriptores de fichero, ya que no pertenecen a la región de ninguno de los procesos.

Descriptor de fichero: punto por el que se escribe en el fichero.

Si se abren dos independientes (en el hijo y en el padre), no comparten nada. Hay dos descriptores distintos. Si se abre uno y se clona, los dos acceden al mismo y puede el padre cambiar el del hijo y el hijo el del padre.

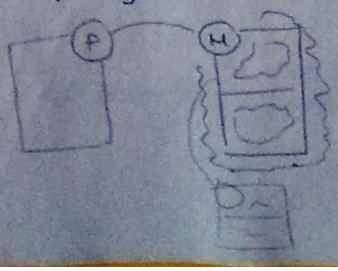
- wait() → espera a la finalización de al menos uno de los hijos

El hijo tiene que "morir" llamando a exit() o return(). Si no se hace, el padre se queda bloqueado: proceso zombie, se queda esperando en wait(). Es un problema.

Si hace fork el padre pero no hace wait, el hijo puede heredar a su abuelo si el padre muere sin hacer wait. No hay ningún problema.

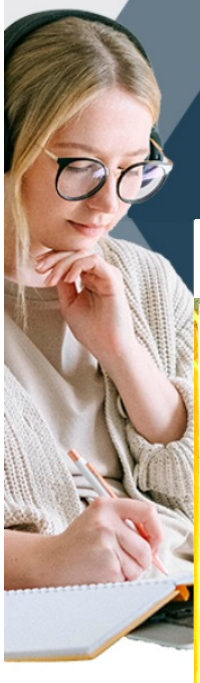
- excep() → contiene una ruta de un programa `excep("/usr/echo", "pepe")`

Realiza una mutación, pierde todas las variables que tenía y muta en el programa que sea p.ej. muta a echo



Son padre-hijo pero ejecutan códigos distintos.

Las líneas de código después de excep() no van a servir.



clona → Inicial → Inicial

muta → Inicial → Memoria

clona → Inicial → memoria, Inicial

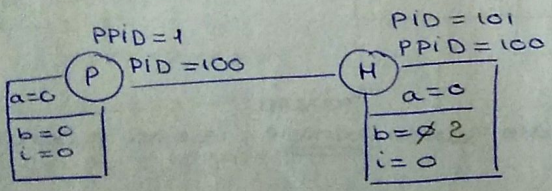
muta → Inicial → Memoria, control.

Es lo que se hace cuando se enciende el ordenador: se clona y se muta para crear el sistema.

Febrero 14

4)

```
int a=0  ← variables globales
int main() {
    pid_t pid;
    int b=0;          } variables locales
    int i, ret;
    for (i=0; i<3; i++)
        if ((pid = fork())==0) {
Hijo {      a = a+2;
            b = b+2;
            ret = execp("echo", "echo", "Proceso", (char *) 0);
            printf(getpid(), getppid(), i, a, b); ← no se ejecuta!!
        }
        else {  ← si hubiese un printf() también habría concurrencia
Padre {     wait(NULL);
            a++;
            b++;
            printf(getpid(), getppid(), i, a, b);
        }
    }
}
```



El padre tiene wait()
⇒ no hay concurrencia

El hijo llama a execlp(), damos por supuesto que eso hace return o exit y muere.

Vuelve el padre ⇒ a=1, b=1, i=1

Genera otro clon...

PANTALLA

El hijo imprime

padre: Proceso 100 1 0 11

hijo: Proceso 100 1 1 22

padre: Proceso 100 1 2 33

Si no hubiese wait()
⇒ concurrencia

Entra primero el elegido por el planificador, se imprime lo mismo pero en distinto orden.

Febrero 14 : Ficheros y procesos

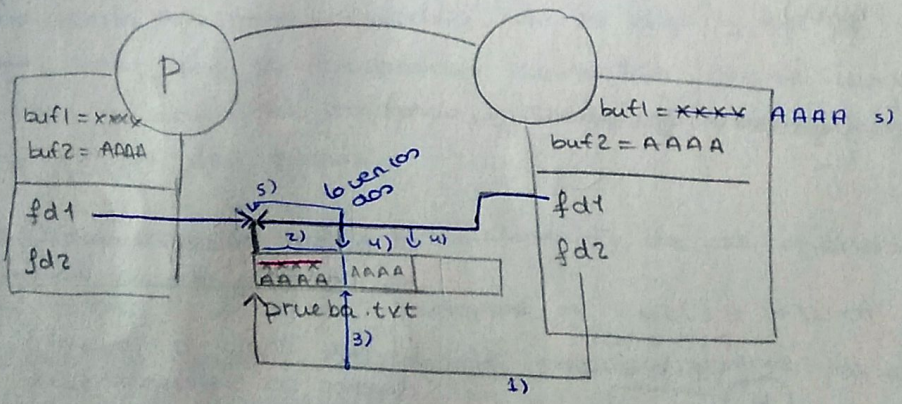
5)

```
char buf1[5] = "xxxx"
char buf2[5] = "AAAA";
int main () {
    int fd1, fd2;
    fd1 = open ("prueba", O_RDWR|O_CREAT |O_TRUNC, 0666);
    if (fork () == 0) {
        fd2 = open ("prueba", O_RDWR);
        write (fd1, buf1, 4); ← el padre lo ve
        write (fd2, buf2, 4); ← el padre no lo ve
        lseek (fd1, 0, SEEK_SET);
        read (fd1, buf1, 4);
        close (fd2);
        close (fd1);
    } Hijo
    else {
        wait (NULL);
        write (fd1, buf2, 4); 4)
        read (fd1, buf1, 4); 6)
        close (fd1);
    } Padre
}
```

El padre abre el fichero con fd1 y hace fork → clonación.

Hay wait() ⇒ primero el hijo.

1) El hijo abre un nuevo descriptor de fichero.
2) El hijo escribe con fd1 (xxxx)
3) " " " con fd2 ¡SOBREESCRIBE! (AAAA) y se desplaza.
5) Leo y desplato al inicio



Problema:

No hay return ni exit, el proceso padre queda zombie, le está esperando con wait.

Si se despertara: 4) escribe AAAA y desplaza 6) Límite del fichero, no escribe nada.

Febrero 15 8)

```
char buf [11] = "xxxxxxxxxx"
int main() {
    int fd1, fd2;
    fd1 = open ("prueba", O_WRONLY | O_CREAT | O_TRUNC, 0666);
    fd2 = open ("prueba", O_RDONLY);
    if (fork() == 0) {
        close (fd2);
        write (fd1, "AAAAA", 5);
        close (fd1);
    }
    else {
        close (fd1); ← concurrente
        wait (NULL);
        read (fd2, buf, 11); ← guarda las 5 A's
        buf[10] = '\0';
        execlp ("/bin/echo", "/bin/echo", buf, NULL);
    }
    printf ("%s\n", buf);
    return 0;
}
```

Hijo

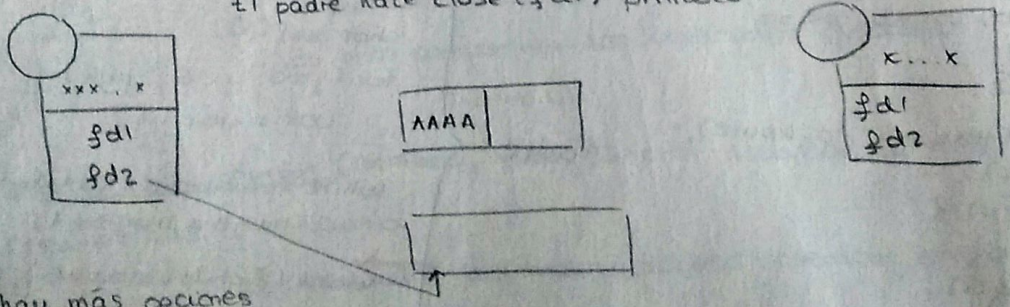
Hijo

Padre

Hijo (el padre hace execlp

¿Es correcto? ¿Hay varias posibilidades de lo que se ve por consola? Correcto.

El padre hace close(fd1) primero



No hay más opciones alternativas.

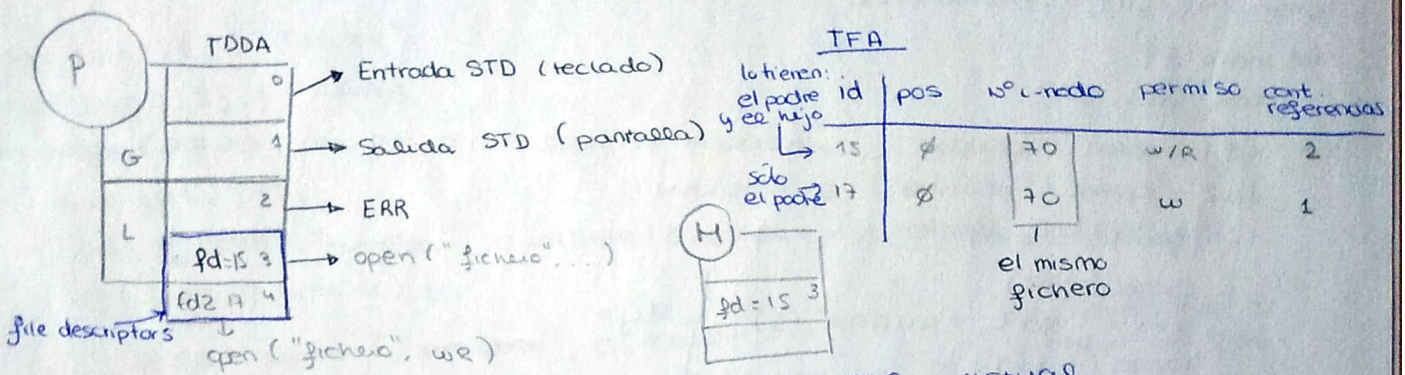
PH → xxxxxxxxxx

PP → AAAAAxxxxx

buf: AAAAAxxxxx

Habría otras si no hubiera wait()

Procesos y ficheros (i-nodos)



V-nodo: i-nodo virtual, en vez de en disco está en memoria virtual.

Tabla de v-nodos

v nodos	contador de referencias
70	3 ← suma de todos los de la TFA de ese i-nodo (tres accesos a ese i-nodo)

Cuando se clona, aumenta el contador de referencias. y ambos tienen acceso a la misma línea de la tabla TFA.

Febrero 17

5)

```
int main () {
    int child_status;
    int numB=0, var=0, pid=0;
    char c;
    int fd1, fd2;
    fd1=open ("text.txt", O_RDWR);
    pid = fork();
    if (pid == 0) {
        int j;
        char buf[1];
        fd2 = open ("text.txt", O_RDONLY);
        for (j=0; j<14 ; j++)
            var = var+1
        while ((c = read (fd2, buf, 1)) != 0)
            numB = numB + 1;

        close (fd2);
        exit(0);
    }
```

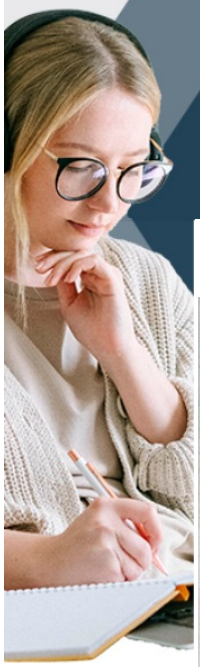
Hijo

```
    else if (pid > 0) {
        int j;
        char buf[1];
        for (j=0; j<6 ; j++)
            var = var + 1;

        while ((c=read(fd1, buf, 1))!=0)
            numB = numB + 1;
        wait (&child_status);
        close(fd1);
        exit(0);
    }
    else {
        fprintf (stderr, "can't fork,
        error %d \n", errno);
        exit (EXIT_FAILURE);
    }
    return 0;
}
```

Padre

Hay concurrencia por dónde está el wait pero no hay problemas porque cada uno hace sus cosas.



a) ¿Qué valor tiene la variable var después de fork()? var = 0

c) ¿Qué valor tiene la variable numB del padre y del hijo después de ejecutarse el padre? Depende del nº de palabras que hubiera en el fichero.

b) Completar tablas. Hijo y padre

TDDA Padre

DF	Ind. TFA
0	230
1	564
2	28
3	12
4	14

TDDA Hijo

DF	Ind. TFA
0	230
1	564
2	28
3	12
4	13

Tabla intermedia de pos (TFA)

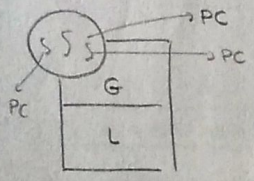
	POS L/E	Nºnodo-i	Perm	Cont.refs
12	Ø EOF	57	W/R	2
13	Ø EOF	57	R	1
14				

Tabla nodos-i

Nodo-i	Cont
57	2

Hilos.

Todo proceso de base es monohilo. Todo hilo tiene asociada un PC.



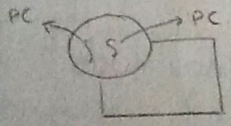
Hilo: funcionalidad asociada al proceso.

Cada hilo tiene un PC asociado.

Los hilos comparten las variables globales.

- pthread_create (nombre, NULL, función asociada al hilo, parámetros función asociada al hilo)
  (NULL: tipo de hilo)

Crea el hilo y lo lanza.



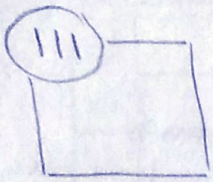
Los dos hilos del mismo proceso intentan entrar a la CPU ⇒ concurrencia.

- pthread_join (nombre_hilo) → espera a que termine ese hilo.

Es bloqueante.

Septiembre 14

6) 1 Concurrencial



a) El hilo 2 nunca sale de bucle while inicial. FALSO

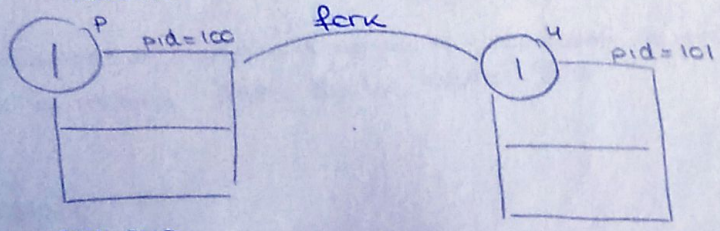
b) Escritura del hilo 2 produce error FALSO

c) Contenido final "aaaa" o bien "bbbb" ninguna otra. FALSO

d) La llamada a close no debería devolver -1. VERDADERO.

Febrero 15

7) Tenemos



con sus variables

Hijo:

A[i] = 0 1 2 3 4 5 6 7

arg1 arg2

Padre:

A [i] = nada

Genera los hilos → th1, th2 cada uno con una mitad de A.

th1. suma [0,3] → 0 1 2 3 = 6

th2. suma [4,7] → 4 5 6 7 = 22

PANTALLA:

101 100 6+22 = 28 (hijo)

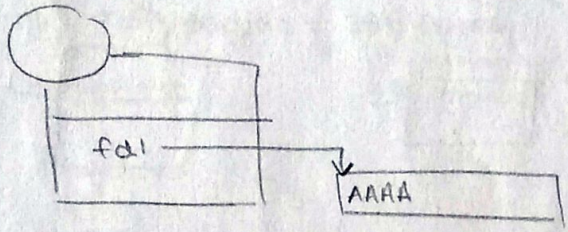
100 1 0+0 = 0 (padre)

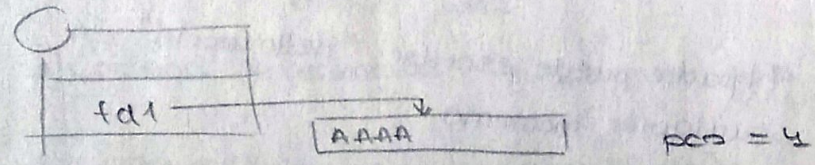
¿Concurrentes como máximo en el sistema?

Como mucho hay dos hilos en el sistema.

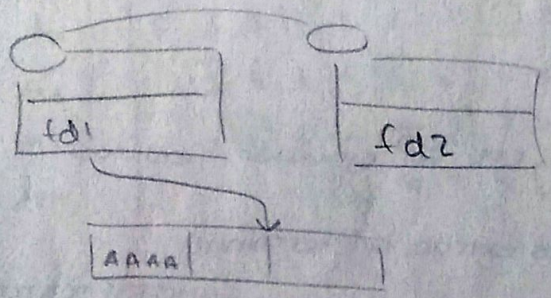
Septiembre 15

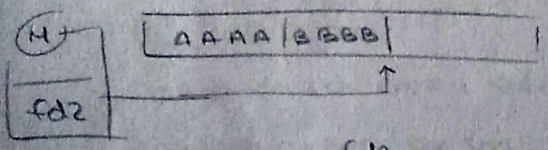
8)

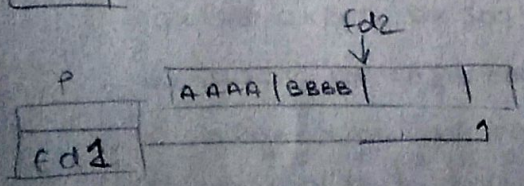


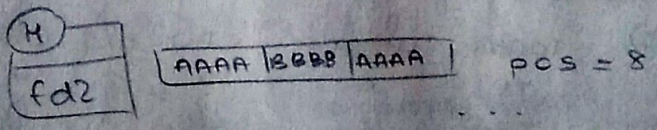


fork()









...

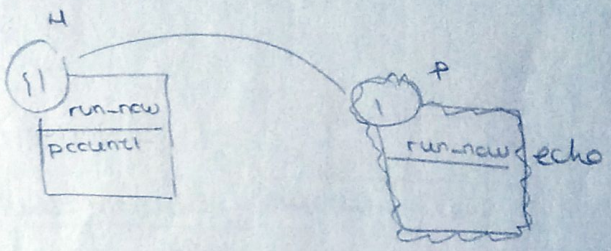
Pantalla: (El padre imprime)

AAAA BBBB AAAA BBBB AAAA BBBB AAAA BBBB

Febrero 16

7)

run_now → se va cambiando para turnarse



Hello Word (padre) | se imprime en cualquier momento

1 (hijo)
2 (hilo)
1 (hijo)
2 (hilo)
1
2
1
2

} Seguro

El padre puede escribir "Hello world" en cualquier momento.

Waiting for thread to finish
Thread joined

Planificador

Expropiativos → extrae procesos de la CPU sin ejecutar enteros y da paso a otros

No expropiativos → da paso y no los extrae sin terminar.

Expropiativo → SJF (más corto primero)
→ RR (raund-robin)
→ quanto 3
→ quanto 2
} nº de ciclos de CPU que puede ej

No expropiativo → SJF (más corto primero)
→ FIFO (FCFS)

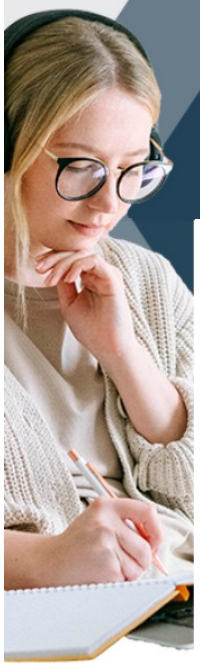
Septiembre 16

2)

Tarea	Llegada	CPU	E/S	CPU
T1	0	4	2	2
T2	2	5	1	1
T3	1	2	3	3
T4	0	6		

a] ¿Última tarea en completar con SJF?

b] Tiempo con RR y q = 3



Memoria

Febrero 16

⑥ Tam página = 256 Bytes = 2^8

0x3A0 0x1B8 0x200 0x320
0x40C 0x6F8 0x504 0x322
0x12C 0x43A 0x380 0x602
0x502

0x3A0 → 0011 | 1010 0000
bloque pal

a) Cadena referencias: 3, 1, 2, 3, 4, 6, 5, 3, 1, 4, 3, 6, 5

b) LRU (último recientemente usado)

#pag	F 3	F 1	F 2	A 3	F 4	F 6	F 5	A 3	F 1	F 4	A 3	F 6	F 5
M1	3	3	3	[3]	3	3	3	[3]	3	3	[3]	3	3
M2		[1]	1	1	1	[6]	6	6	6	[4]	4	4	4
M3			[2]	2	2	2	[5]	5	5	5	5	[6]	6
M4					[4]	4	4	4	[1]	1	1	1	[5]

Febrero 15

⑥ 256 Bytes

0x10 0x31A 0xF4 0x17C 0x3B8 0x284 0x4FE 0x4C 0x334
0x158 0x26D 0x502

a) Cadena de referencia: 0, 3, 0, 1, 3, 2, 4, 0, 3, 1, 2, 5

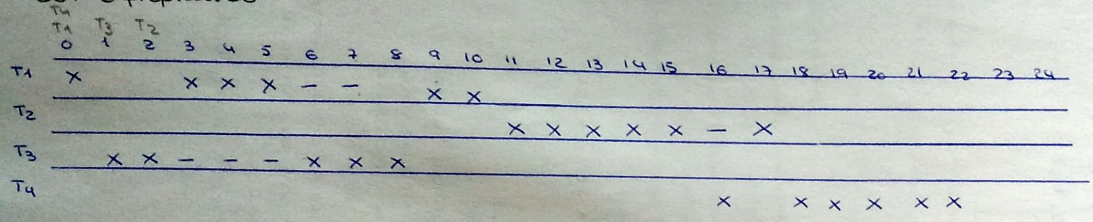
b) Algoritmo del reloj

#pag	F 0	F 3	A 0	F 1	A 3	F 2	F 4	A 0	A 3	F 1	F 2	F 5
M1	0	0	0¹	0¹	0¹	0¹	0	0¹	0¹	0¹	0	0
M2	[ ]	3	3	3	3¹	3¹	3	3	3¹	3¹	3	3
M3		[ ]	[ ]	1	1	1	4	4	4	4	2	2
M4				[ ]	[ ]	2	[2]	[2]	[2]	1	[1]	5

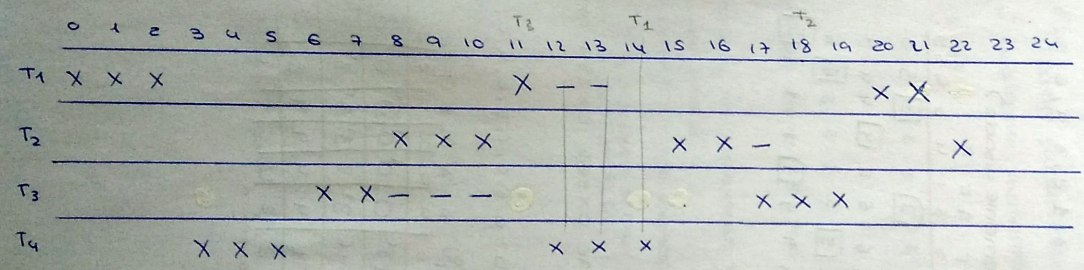
candidato a salir

Se ganan "vidas" que impiden que sea candidato a salir.
Pierde la vida cuando pasan su turno de salir.
Acierto → se gana vida

SJF expropiativo



Rand-robin q=3



T1 T4 T3 T2 T1 T4 T2 T3 T1 T2

Febrero 2016

④ 16 KB para datos (tam disco) 1KB: tam bloque

FAT

nombre	tipo	fecha	tam (bytes)	1er bloque
tasix.tex	FR	01/01/16	5567	3
makefile	FR	03/1/16	1048	0
escudo.png	FR	02/1/16	4280	6

1KB = 1024 B

a) Posible contenido de la tabla FAT.

FAT

0	1	8	EOF
1	EOF	9	10
2	4	10	11
3	2	11	12
4	5	12	EOF
5	7	13	
6	9	14	
7	8	15	

$\frac{5567}{1024} = 5'43 \simeq 6$ bloques

$\frac{1048}{1024} = 2$ bloques

$\frac{4280}{1024} = 4'17 \simeq 5$ bloques

b) Enlace rígido a tasix.tex, incluyendo tesitlist. FR 02/01/16 tamaño 5567 1er bloque

¿Es realmente un enlace rígido?

En FAT no hay enlaces rígidos, ~~[illegible]~~ es una cosa de i-nodos y linux.

Septiembre 2016

④

	root	home	usr	febrero.odt	septiembre.txt	junio.odt	tmp.txt	carpeta
nodo-i	2	4	7	8	9	10	11	16
tipo	D	D	D	F	F	F	F	D
D1	1	2	4	5	10	15	9	3
D2				6	12	8		
I								

m.bits

1111101111101001

00110

7 → esto no, ocupa dos bloques 18 y 19

root 1

.	2
..	2
home	4
usr	7

home 2

.	4
..	2
carpeta	16
tmp.txt	11

3

.	16
..	4
febrero.odt	8
septiembre.txt	9
junio.odt	10

4 usr

.	7
..	2

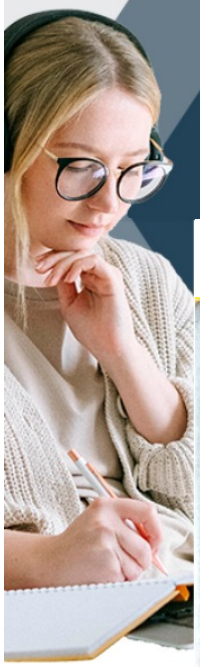
7

18
19

Tabla FAT

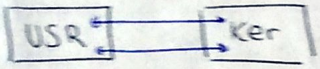
Blq	Índice	Blq	Índice
1	<EOF>	11	<EOF>
2	<EOF>	12	<EOF>
3	<EOF>	13	
4	<EOF>	14	
5	6	15	8
6	<EOF>	16	
7		17	
8	18	18	19
9	<EOF>	19	EOF
10	12	20	

Nombre	tipo	nº bloque	
root	D	1	
home	D	2	
usr	D	4	
carpeta	D	3	
tmp.txt	F	9	
febrero.odt	F	5	→ dos bloques
junio.odt	F	15	→ ocupa 4 bloques
sept.odt	F	12	



Drivers

/dev (ratón, impresora ...) { open, read CAT >, write ECHO >, release



Mayor → id de una familia de dispositivos

Minor → id de dispositivo (va ligado a un mayor)

Comando:

sudo mknod -c 250 0
- -c: fichero de tipo carácter
- 250: mayor
- 0: minor

Septiembre 17

② Indique que afirmaciones son verdaderas y cuales falsas

a) Modulo de kernel accede a un...

Falso El kernel no descarga módulos

b) Dos drivers activos mismo major...

Falso Dos drivers no pueden tener el mismo major, porque por eso se tiene el major, para diferenciar drivers

c) Desde kernel podemos usar librerías estandar de C. Falso

No, no se puede. Puedes incluir /linux, pero librerías de C

d) Cada vez que se carga el kernel... verdadero

Febrero 2017

① Bloques de 4kB $= 2^{12}$ Punteros de 64 bits 0 → 0 1 → I 1 → I^2

Programar en C. Crear fichero (si ya existe trunca) escribir posicionamiento y lectura bloque 10.

1) Crear fichero directorio trabajo actual.

```
int fd = open ("outfile", O_CREAT | O_TRUNC | O_RDWR, 0666)
```

2) Escritura en 11 bloques (del 0 al 10)

Buffer: tamaño de un bloque

```
char buff [4096] ;= // tamaño de un bloque
                 }of
int i
for (i=0; i<11; i++) {
    write (fd, buff, 4096)
}
```

3) Posicionamiento al comienzo del bloque 10 y lectura del bloque 10.

opción A

```
lseek (fd, -4096, SEEK_CUR)
```

opción B

```
eseek (fd, 40960, SEEK_SET)
```

```
read (fd, buff, 4096)
```

4) Posicionamiento al principio y lectura

```
eseek (fd, 0, SEEK_SET)
int i;
for (i=0; i<10; i++) {
    read (fd, buff, 4096)
}
```

5) Posicionamiento en 3051270 y escritura "fin de fichero"

lseek (fd, 3051270, SEEK_SET) → ¡no puede ser! No están creados esos bloques

Calcular cuántos bloques hay ahí.

$\frac{3051270}{4096} = 744'9 \simeq 745$ bloques que tengo que escribir

```
lseek (fd, 0, SEEK_SET)
int i;
for (i=0; i<745; i++)
    write (fd, buff, 4096)
lseek (fd, 3051720, SEEK_SET)
write (fd, "fin de fichero", 14)
```

b) Nivel de punteros del 3051270 (directo, indirecto o indirecto^2)

Cuantos punteritos tenemos en un indirecto.

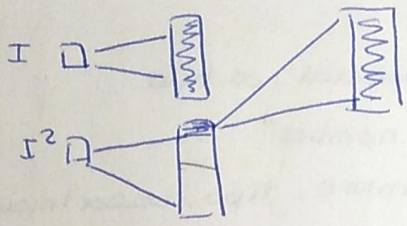
4KBytes $= 2^{10} \cdot 2^2 = 2^{12}$

64 bits $= \frac{64}{8} = 8 = 2^3$

$\frac{2^{12}}{2^3} = 2^9$ punteritos $= 512$ en indirecto

D 10
I 512 +
522

745 > 522 ⇒ nivel indirecto doble.



c) Tam. máx fichero.

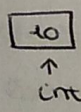
$(10 + 512 + 512^2) \times 4KB \simeq 1GB$.

KB → 2^{10}
MB → 2^{20}
GB → 2^{30}

Septiembre 2017

① TFA a) Tamaño máximo fichero.

ind Puntero nºinodo



Cada puntero son 4 Bytes.

Tengo 0 ... hasta el que limite del entero.

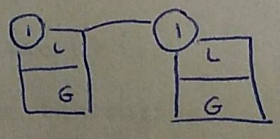
4B son 8 bits → $8 \times 4 = 32$ bits

2^{32} 1...1 (32 veces) $2^{30} \cdot 2^2 = \boxed{4GB}$

$\frac{32}{8} = 2^2$

↳ tam máx teniendo en cuenta los bloques $\frac{2^{11}}{2^2} = 2^9$

¿Y el de la partición? (D, I, I^2) $\simeq 1GB$ $(10 + 2^9 + 2^{9^2}) \times 2KB = 1GB$

b) 

¿Parentesco?

Sí, el proceso 2 y el proceso 3 están emparentados. (los 3) mismo FD

No hay forma de saber quién es padre o hijo.

c) P1 hace close(3)

Desaparece toda la línea del índice 4 del TFA y #Regs 1

De la TDDA P1 quitamos FD 3.

d) P1 open ("/tmp/file.txt", O_RDWR)

Utilizamos el FD3, que hemos quitado antes, IDFF 4

y en la TFA añadimos 4 0 98 r/w 1

Febrero 2015

② retirarPizzaBuffet (tipoPizza) peperoni (0) margarita (1)

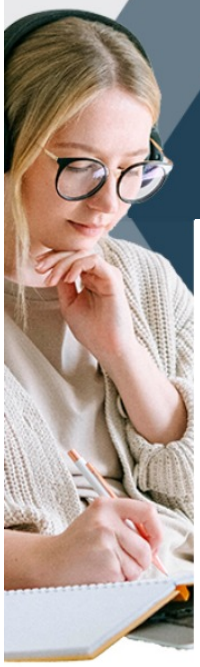
reponerPizzas (N, tipoPizza)

```
void Cliente (int tipoPizza) {
    while (true) {
        conseguirPizza (tipoPizza); ← implementan
        comer ();
    }
}

void Camarero () {
    while (true) {
        servirPizzas (); ← implementan
    }
}

mutex_t cerrojo;
contador-pizzas [2];          cond_t camarero;
conseguir-pizza (int tipoPizza) {   cond_t tipoPizza [cond [2]
    lock (cerrojo);
  while (contador_pizzas [tipo-pizza] == 0) {
        cond-signal (camarero); ← aviso al camarero de que no hay pizzas
      cond-wait (tipoPizza [cond [tipoPizza], cerrojo) ← si salgo de wait es que hay una pizza para mi
  }
   retirarPizzaBuffet (tipoPizza);
   unlock (cerrojo);

 servirPizza () {
    lock (cerrojo);
   cond_wait (camarero, cerrojo); ← si salgo del wait es porque me han
    if (contador-pizzas [0] == 0) { avisado de que no hay pizza
        reponerPizzas (N, 0);
        cond-broadcast (tipoPizzaCond [0]); ← aviso a todos los que esperan una pizza 0
    }
    if (contador-pizzas [1] == 0) {
        reponerPizzas (N, 1)
        cond_broadcast (tipoPizzaCond [1]);
    } unlock (cerrojo); }
```



Septiembre 2016

② e e e e
0 1 2 3
0 4 8 12
1 5 9 13 } Corredores.
2 6 10 14
3 7 11 15

```
void corredor (int id_corredor) {
    start_race(id_corredor); //bloquea al corredor hasta su turno
    run(id_corredor);
    notify_arrival(id_corredor);
}
```

Todos se bloquearán hasta que los demás hayan invocado a start_race()

```
cond_t equipos[4];          int llegada = 0;
int turno[4];               cond_t llegadaCond;
mutex_t cerrojo;
start_race() { //(barrier)
    lock(cerrojo);
    llegada++;
    if(llegada != 16)  //si no es el último, a dormir
      cond_wait(llegadaCond, cerrojo);
    else  //el último es el que despierta a todos
      cond_broadcast(llegadaCond);

//Ahora, solo tienen que salir 0, 4, 8, 12  (comprobar turno)
//Si no es mi turno me voy a dormir con una condición nueva.
while(turno[id_corredor / 4] != id_corredor % 4) {   (equipo)
      cond_wait(equipos[id_corredor/4], cerrojo);
}
unlock(cerrojo);
}
notify_arrival(idcorredor) {
    lock(cerrojo);
    if(id_corredor % 4 != 3) {
      turno[id_corredor/4]++;
      cond_broadcast(equipos[id_corredor/4]);
    }
    else { printf("%d. %d", id_corredor/4, posicionmeta)
           posicionmeta++; } unlock(cerrojo); }
```

cuando despierta al equipo, tienen todos que revaluar su condición, para ver si les toca salir o no. Si no les toca vuelven a dormir.

Septiembre 2016

③

1) /dev

2) major minor

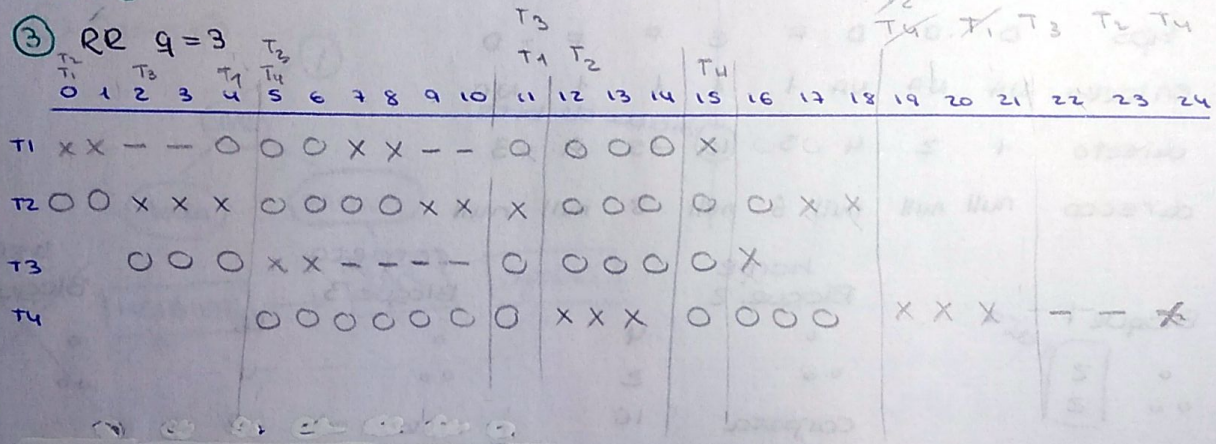
3) ~~make file~~

make

Sudo insmod leds.ko

Febrero 17

③ RR q=3



T1 → 16 → 5 x, 4 -, 7 o tiempo espera: 7/16

T2 → 14/19 T3 → 8/15 Tu → 11/20

Porcentaje de uso:

2 parada en 25 ciclos. $100 - \frac{2}{25} \cdot 100 \rightarrow 92\%$

$$\text{Productividad} = \frac{\text{nº tareas}}{\frac{\text{nº ciclos uso}}{\text{ciclos totales}}} = \frac{4}{23/25}$$

Febrero 17

```
mkdir usr
cd home/carpeta
ln nada.odt cosas.odt
fn -s ../archivo.txt acceso
rm nada.odt
```

mapa de bits

1 1 1 (1) 1 1 0 1 1 [1] 0 0 0 0 1 0

(1) → 0 mkdir usr

[1] → 0 ln -s

a) Contenido de la tabla de nodos-i, bloques relevantes y mapa de bits

nodo-i	2 (raíz)	4 (home)	7 (usr)	8 (valores)	9 (acceso)	10 (nada cosas)	11 (archivo)	16 (carpeta)
tipo	D	D	D	F	E	F	F	D
enlaces	NA	NA	NA	1	1	1	1	NA
directo	1	2	4	5	(10)	15	9	3
directo	null	null	null	6	null	8	null	null

mapa de bits

Bloque 1 (raíz)

.	2
..	2
home	4
~~usr~~	~~7~~

home Bloque 2

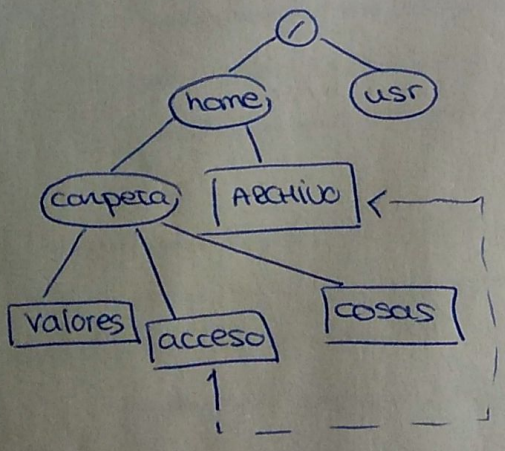
.	4
..	2
carpeta	16
archivo.txt	11

carpeta Bloque 3

.	16
..	4
valores.odt	8
~~acceso~~	~~9~~
~~cosas.odt~~	~~10~~
nada.odt	10

usr Bloque 4

.	7
..	2



Deshacer de abajo a arriba.

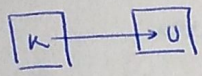
- rm nada.odt ¿qué era nada? miramos arriba → cd home/carpeta, estaba ahí.
- ln -s ../archivo.txt acceso
  quitamos acceso. (mapa bits 9)
- ln nada.odt cosas.odt
  Quitamos cosas.odt
- mkdir usr (mapa bits 4)

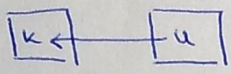
Febrero 2013

(3)

device_read

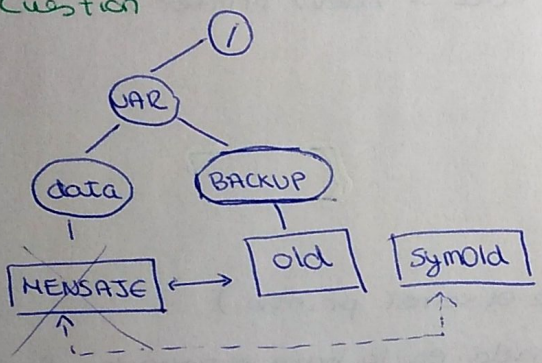
↓

copy_to_user  coger algo del kernel y pasarlo al usuario.

copy_from_user 

Febrero 2015

(1) Cuestión



a) Comandos para hacer esto.

mkdir /var/backup (1)

en /var/data/mensaje /var/backup/old

en -s /var/data/mensaje /var/backup/symOld. (2)

rm /var/data/mensaje

(b) ¿cuántos nodos-i nuevos?

Uno para el directorio, otro para el enlace simbólico = 2

No se destruye ninguno porque hay un enlace rígido.

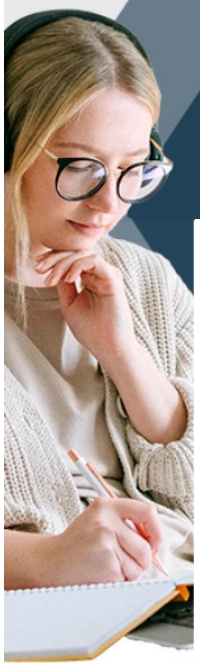
c) ¿Qué se muestra por pantalla?

> cat /var/backup/old

3+2 != 1+4

> cat /var/backup/symold

ERROR



Febrero 2015

(2) release → eliminar.

a) mknod /dev/printer c 249 0
- mknod: crear módulo
- /dev/printer: ruta donde se va a crear el dispositivo
- c: tipo caracter
- 249: major
- 0: minor

b) echo "My printer is working!" > /dev/printer

echo → escritura en el dispositivo (write)

Si fuera cat → lectura (cat > /dev/printer)

Febrero 18

(3)

a) Falso. Es el kernel.

b) Verdadero. (Imprimir desde el kernel printk).

c) Falso. El kernel no descarga nada, eso lo hace el usuario con rmod.

d) Falso, dos drivers no pueden tener el mismo major. insmod para cargar

e) ~~Falso~~. Se compila con un makefile. Verdadero. Con sudo.

Febrero 16

(5)

a) Las constantes no ocupan espacio.

Variable / Macro	Ocupa espacio	Región del mapa de memoria
CONSTANT	~~NO~~ SI (según Marcos)	NO
num2	SI	Variables sin valor inicial
num1	SI	Variables con valor inicial
i	NO	pila sistema
*i	NO	~~NO~~ pila proceso

Septiembre 17

① D 10 I 1 I^2 1 I^3 1 700.010 Bytes.

Tam. bloque 4kB 32 bits puntero.

11

2^{12} Bytes → 4096

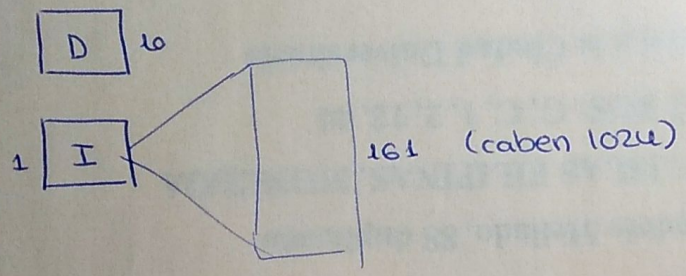
$\frac{700.010}{4096} = 170'9 \simeq 171$ bloques. que ocupa el fichero.

b) En FAT 171 bloques

a) $\frac{2^{12}}{2^2} = 2^{10} = 1024$ punteros contiene cada bloque indirecto.

$32/8 = 4$

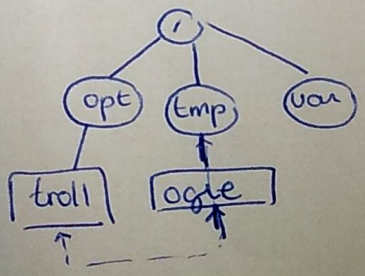
Ocupo 10 directos + 1 indirecto + 161 = 172 bloques físicos



Septiembre 16

② Problema:

nodo-i	2 (/)	3 (opt)	4	5 (tmp)	6 (var)	7	8	9	10
enlaces	NA	NA	1	NA	NA	1			
Tipo	D	D	F	D	D	S			
Tamaño	32	14	70	7	7	10			
D1	6	7	2	8	9	0			
D2	/	/	3	/	/				
I	/	/	4	/	/				



Mapa 00 1 1 1 1 1

0 1 2 3 4 5 6 7

tamaño ogre: 10

tam troll $70/32 = 2'18 \simeq 3$ bloques lógicos.

5

Bloque 6 (/)
. 2
.. 2
opt 3
tmp 5
var 6

Bloque 7 (opt)
. 3
.. 2
troll 4

Bloque 8 (tmp.)
. 5
.. 2
ogre 7
krampes 4

Bloque 9 (var)
. 6
.. 2