

# Ejercicios-SO-resueltos.pdf



beaas\_218



Sistemas Operativos



2º Grado en Ingeniería del Software



Facultad de Informática  
Universidad Complutense de Madrid



Que no te escriban poemas de amor  
cuando terminen la carrera ►►►►►►►

Smiley face icon

(a nosotros por  
suerte nos pasa)

**WUOLAH**



(a nosotros por suerte nos pasa)

No si antes decíte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

## EJERCICIOS SO

### SISTEMAS DE FICHEROS

1. Un dispositivo de memoria flash de 64 MB de capacidad y bloques de 1 KiB, contiene un sistema de ficheros FAT.

- a) ¿Cuántos bytes son necesarios para almacenar la tabla FAT?
- a. 64 KiB
  - b. 128 KiB
  - c. 1 MiB
  - d. 512 KiB
  - e. Ninguna de las respuestas anteriores es correcta

La capacidad de la memoria es de 64MB =  $2^{26}$  bytes, y el tamaño de bloque es de 1KB =  $2^{10}$  bytes, por lo que el número de bloques será

$(2^{26} \text{ bytes}) / (2^{10} \text{ bytes/bloque}) = 2^{16}$  bloques. Por lo tanto, como la tabla FAT tiene una entrada por cada bloque, tendrá  $2^{16}$  entradas. Si suponemos un sistema basado en FAT-32 (es decir, que cada entrada está codificada con 32 bits = 4 bytes), el tamaño de la tabla FAT será  $(2^{16} \text{ entradas}) * (2^2 \text{ bytes/entrada}) = 2^{18}$  bytes =  $2^8 \text{ KB} = 256 \text{ KB}$ . Es decir, la respuesta correcta es la e) Ninguna de las anteriores.

b) ¿Es posible realizar enlaces rígidos en un sistema de ficheros tipo FAT?

En un sistema de ficheros tipo FAT el contenido de un directorio es, para cada fichero, su nombre, tamaño, otros atributos y el número del primer bloque físico donde está almacenado el fichero (a diferencia del sistema de nodos-i, donde cada entrada del directorio contiene únicamente el nombre y el número de su nodo-i). Por lo tanto, si quisieramos crear un enlace rígido para el fichero X habría que incluir en el directorio una entrada con un nombre distinto, y con el resto de atributos (tamaño, primer BF, etc) iguales que los de X. Ahora bien, esto implica que cada vez que modificásemos ese fichero (cambiando su tamaño, o incluso el BF donde empieza) tendríamos que actualizar los atributos en todos los posibles enlaces rígidos que se hayan creado. Esto es inviable, ya que además habría que buscar por todo el disco para encontrar todos los enlaces. Por tanto, **no** se pueden realizar enlaces rígidos en un sistema de ficheros tipo FAT.

c) ¿Por qué no se puede establecer enlaces rígidos a ficheros de volúmenes distintos en sistemas de ficheros tipo ext2?

En un sistema de ficheros que utiliza los nodos-i, en el directorio se almacena el nombre del fichero y su número de nodo-i. Sin embargo, cada volumen tiene su propia tabla de nodos-i, por lo que un enlace rígido de un volumen que tenga el número de nodo-i '4', por ejemplo, hará referencia al nodo-i 4 de ese volumen, no de otro.

2. Un sistema de ficheros UNIX utiliza bloques de 1024 bytes y direcciones de disco de 16 bits. Los nodos-i (entradas en una tabla que contiene la información descriptiva de los ficheros) contienen 8 direcciones de disco para bloques de datos, una dirección de bloque índice indirecto simple y una dirección de bloque índice indirecto doble. Para indicar el tamaño del fichero y el desplazamiento, (**offset**) de la posición en bytes en las operaciones *read* y *write*, se utilizan números de 32bits.

a) ¿Cuál es el tamaño máximo de un fichero en este sistema? ¿y el de la partición?

Tenemos los siguientes datos:

- Tamaño de bloque = 1024 bytes =  $2^{10}$  bytes
- Dirección de disco de 16 bits = 2 bytes
- Cada nodo-i tiene { 8 punteros directos
  - { 1 puntero indirecto simple
  - { 1 puntero indirecto doble
- Para el offset se necesitan 32 bits = 4 bytes

Para determinar el tamaño máximo de un fichero tenemos que analizar 3 cuestiones:

1. Número máximo de bloques:  $8 + (2^{10})/2 + ((2^{10})/2)^2$  bloques, que equivalen a  $(8 + (2^{10})/2 + ((2^{10})/2)^2) * (2^{10} \text{ bytes/bloque}) = 2^{13} + 2^{19} + 2^{28}$  bytes
- Los primeros 8 bloques corresponden a los referenciados por los punteros directos.
- El puntero indirecto simple apuntará a un bloque de  $2^{10}$  bytes, donde cada entrada de 2 bytes será un puntero a otro bloque. Se podrán referenciar por tanto  $2^{10}/2$  bloques

- El puntero indirecto doble será un puntero a un bloque de  $2^{10}$  bytes, donde cada entrada de 2 bytes será un puntero a otro bloque. En este primer bloque se podrán referenciar por tanto  $2^{10}/2$  bloques. Ahora bien, cada uno de los  $2^{10}$  bloques referenciados por el primero tendrá a su vez  $2^{10}/2$  punteros a bloques, por lo que el número total de bloques referenciados como máximo será  $((2^{10})/2) * ((2^{10})/2) = ((2^{10})/2)^2$
- 2. Tamaño de las direcciones del disco: con 16 bits para las direcciones del disco se pueden referenciar hasta  $2^{16}$  bloques, que equivalen a  $(2^{16} \text{ bloques}) * (2^{10} \text{ bytes/bloque}) = 2^{26} \text{ bytes}$
- 3. Offset: con un desplazamiento codificado con 32 bits podemos referenciar hasta  $2^{32}$  bytes

Lo que limita entonces el tamaño del fichero es el tamaño de la dirección del disco (los 16 bits), es decir, podrá tener un tamaño máximo de  **$2^{26}$  bytes**

b) ¿Qué habría que cambiar para aumentar el tamaño del fichero? (tamaño de bloque, puntero, incluir indirecto triple...)

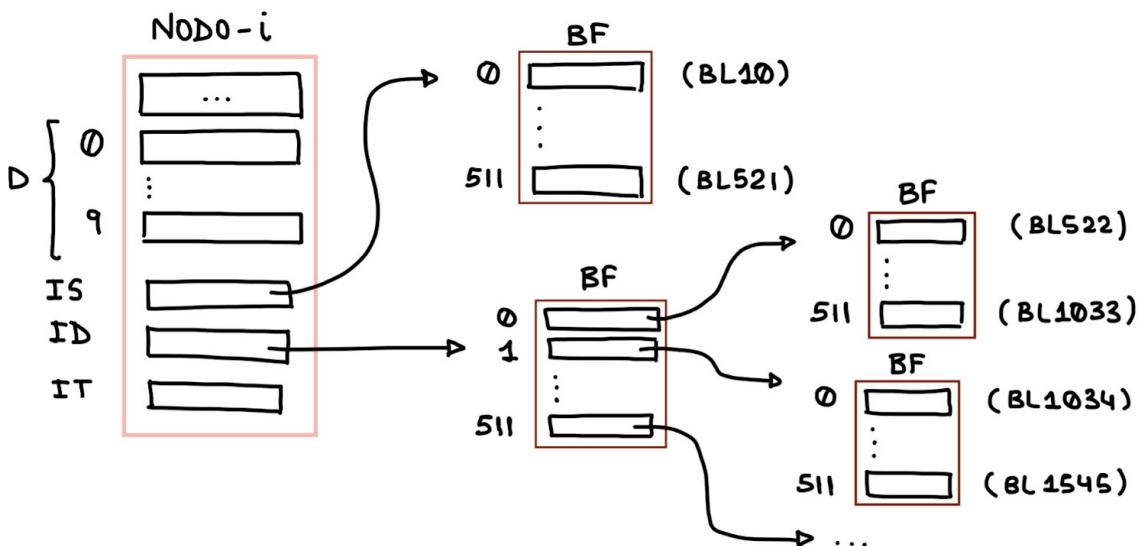
En este caso se podría aumentar el tamaño de la dirección del disco, ya que es lo que más limita su tamaño.

3. Un programa UNIX crea un fichero e inmediatamente se posiciona (con lseek) en el byte 55 millones. Luego escribe un byte. ¿Cuántos bloques de disco ocupa ahora el fichero (incluyendo bloques indirectos)? Asúmase que los nodos-i contienen 10 índices directos, 1 índice indirecto simple, 1 índice indirecto doble y 1 índice indirecto triple, los bloques tienen un tamaño de 2 Kbytes y el tamaño de los índices (direcciones de bloques de disco) es de 32 bits. ¿Qué sucesión de índices lógicos nos lleva al byte posicionado por lseek?

Tenemos los siguientes datos:

- Tamaño de bloque: 2KB =  $2^{11}$  bytes
- Dirección de disco: 32 bits = 4 bytes =  $2^2$  bytes
- Nodo-i: { 10 punteros directos
  - { 1 puntero indirecto simple
  - { 1 puntero indirecto doble
  - { 1 puntero indirecto triple

El byte 55 millones equivale al BL **26.855** ( $55.000.000 \text{ bytes}) / (2^{11} \text{ bytes/bloque})$



- En los punteros directos 0-9 encontramos los números de BF donde están los BL 0-9. Aquí no encontraremos entonces el BL 26.855
- El puntero indirecto simple tiene el número de un BF de  $2^9$  entradas ( $(2^{11} \text{ bytes/bloque}) / (2^2 \text{ bytes/dirección})$ ), cada una apuntando a un BF (el puntero 0 apuntará al BF que tenga el contenido del BL10, el puntero 1 al BF que contenga el bloque BL11, etc.). Por lo tanto, a partir del puntero indirecto simple podemos acceder a los 512 bloques físicos que tendrán el contenido de los bloques lógicos BL10-BL521. Aquí tampoco encontraremos entonces el BL 26.855
- El puntero indirecto doble tiene el número de un BF de 512 entradas (mismo razonamiento), cada una apuntando a otro BF con 512 entradas, que a su vez apuntarán a los BF donde estén los BL correspondientes. Por

# Que no te escriban poemas de amor cuando terminen la carrera

(a nosotros por suerte nos pasa)



Ayer a las 20:20

Oh Wuolah wuolitah  
Tu que eres tan bonita

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

No si antes decirte  
Lo mucho que te voy a recordar



Envía un mensaje...



**WUOLAH**

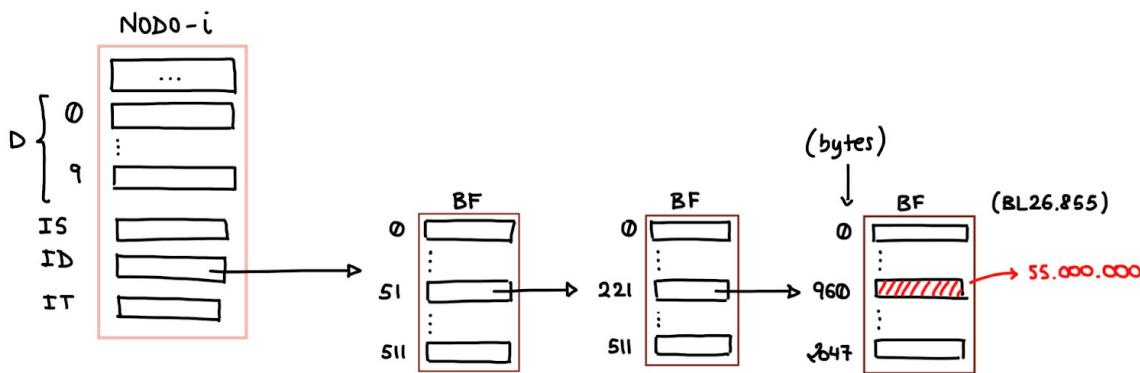


lo tanto, a partir del puntero indirecto doble podemos acceder a  $(2^9) * (2^9) = (2^9)^2 = 2^{18} = 262.144$ . Con este puntero indirecto doble podremos acceder al bloque que necesitamos

A partir de una entrada  $e$  del BF del primer nivel podemos acceder al BL  
 $e * 512 + 10 + 512$  (y hasta el BL  $e * 512 + 10 + 512 + 512 - 1$ ). Por lo tanto, para acceder al contenido del BL 26.855 basta calcular:  
 $e * 512 + 10 + 512 = 26.855 \Leftrightarrow e = (26.855 - 10 - 512) / 512 = 51,43$ , es decir, a la entrada **51** del BF del primer nivel.

Esa entrada apuntará al BF cuyas entradas apuntarán a su vez a los BF que contendrán los BL desde el 26.634 ( $51 * 512 + 10 + 512$ ) hasta el 27.145 ( $26.634 + 511$ ). Es decir, de este BF del segundo nivel habrá que acceder a la entrada **221** ( $26.855 - 26.634$ )

El camino para llegar al BL 26.855 será:



Dentro del BF que contiene al BL 26.855 habrá que leer el byte **960**, ya que el primer byte del BL 26.855 es el 54.999.040 ( $26.855 * 2^{11}$ ), y por tanto el 55.000.000 será el 960 ( $55.000.000 - 54.999.040$ )

En cuanto al tamaño, este fichero sólo ocupará **3** bloques de disco (l2 indirectos y el que contiene el byte que se ha escrito).

**4.** Juan crea un fichero de nombre *JFichero*. María crea un enlace físico a *JFichero* y le da el nombre *MFichero*. Juan elimina *JFichero*. Luego Juan crea un nuevo fichero y también le llama *JFichero*.

a) ¿Cuántos ficheros diferentes existen después de las acciones anteriores?

Existen 2 ficheros diferentes (cuando Juan elimina *JFichero* sólo se está decrementando su número de enlaces).

b) ¿Sería diferente la respuesta si el enlace que ha creado María fuese un enlace simbólico de nombre *MFichero* que apuntara a *JFichero*?

Si fuese un enlace simbólico sólo existirían también 2 ficheros, el último *JFichero* creado por Juan, y el fichero *MFichero* de tipo enlace que apuntaría a ese *JFichero*.

**5.** El sistema de ficheros de un SO diseñado a partir de UNIX utiliza bloques de disco de 1024 bytes de capacidad. Para el direccionamiento de estos bloques se utilizan punteros de 16 bits. Para indicar el tamaño del fichero y el desplazamiento (*offset*) de la posición en bytes en las operaciones *read* y *write*, se utilizan números de 64 bits. Cada nodo-i tiene 8 punteros de direccionamiento directo, 1 puntero indirecto simple y 1 puntero indirecto doble.

a) ¿Cuál será el tamaño máximo de un fichero suponiendo despreciable el espacio ocupado por el superbloque y la tabla de nodos-i?

Al igual que en el ejercicio 4, analizamos 3 posibles limitaciones del tamaño del fichero:

- Número máximo de bloques:

$$(2^3 + (2^{10}/2) + (2^{10}/2)^2) \text{ bloques} * 2^{10} \text{ bytes/bloque} = \\ = 2^{13} + 2^{19} + 2^{28} \text{ bytes}$$

2. Tamaño de las direcciones del disco: con 16 bits para el direccionamiento de los bloques se pueden referenciar hasta  $2^{16}$  bloques, que equivalen a  

$$(2^{16} \text{ bloques}) * (2^{10} \text{ bytes/bloque}) = 2^{26} \text{ bytes}$$
3. Offset: con un desplazamiento codificado con 64 bits podemos referenciar hasta  $2^{64}$  bytes

Lo que limita entonces el tamaño del fichero es el tamaño de la dirección del disco (los 16 bits), es decir, podrá tener un tamaño máximo de  $2^{26}$  bytes

**b) Si se modifica el tamaño de puntero pasándolo a 32 bits, ¿cuál será el nuevo tamaño máximo?**

Mismo procedimiento que en el apartado anterior:

1. Número máximo de bloques:  

$$(2^3 + (2^{10}/4) + (2^{10}/4)^2) \text{ bloques} * 2^{10} \text{ bytes/bloque} =$$

$$= 2^{13} + 2^{18} + 2^{26} \text{ bytes}$$
2. Tamaño de las direcciones del disco: ahora con 32 bits podremos referenciar hasta  $2^{32}$  bloques, que equivalen a  $(2^{32} \text{ bloques}) * (2^{10} \text{ bytes/bloque}) = 2^{42} \text{ bytes}$
3. Offset: igual que antes, podemos referenciar hasta  $2^{64}$  bytes

Lo que limita ahora el tamaño del fichero es el número máximo de bloques que podemos referenciar por nuestro sistema de nodos-i, es decir, podrá tener un tamaño máximo de  $2^{13} + 2^{18} + 2^{26}$  bytes = **67379200 bytes** (aprox,  $2^{26}$  bytes)

**6. Sugerir una razón por la cual alguien pudiera desear construir un directorio sobre los cuales los demás usuarios tuvieran permiso de ejecución pero no de lectura, en un sistema de ficheros de tipo UNIX.**

Podría desearse construir un directorio que los demás usuarios pudieran atravesar para acceder a ficheros (u otros directorios) conocidos por ellos, pero queriendo mantener una serie de archivos ocultos o desconocidos para esos usuarios. Así, eliminando los permisos de lectura impediría a dichos usuarios ejecutar comandos como 'ls' para conocer **todos** los ficheros de ese directorio, pero dejando que accedieran a los archivos conocidos manteniendo el permiso de ejecución.

**7. Dos estudiantes de informática, Estudiante1 y Estudiante2, sostienen una discusión respecto a los *nodos-i*. Estudiante1 argumenta que, dado que las memorias son cada vez más grandes y baratas, cuando se abre un fichero es más sencillo y rápido obtener una nueva copia del *nodo-i* para llevarla a la tabla de *nodos-i* en memoria, que buscar en la tabla entera para comprobar si ya está allí. Estudiante2 discrepa. ¿Quién tiene razón?**

Considero que tiene razón el Estudiante2, es decir, no conviene obtener una nueva copia del nodo-i cada vez que se abre un fichero. En primer lugar, el acceso a disco para encontrar y posteriormente copiar el nodo-i es mucho más lento que el acceso a la memoria principal para buscar el nodo-i, por lo que en todos los casos de acierto se estaría desperdimando mucho tiempo. De hecho, comprobar si el nodo-i está en memoria podría ser tan sencillo como saber cuándo una página está en memoria física, es decir, comprobando un bit de validez en la tabla de páginas. Además, el tener varias copias de un mismo nodo-i en memoria puede traer problemas de coherencia. A la hora de sustituir un nodo-i porque no quedan huecos libres, habría que actualizar su contenido en disco antes de eliminarlo de la tabla. Sin embargo, puede que esa copia del nodo-i no sea la más reciente y, por tanto, contenga datos incoherentes o desactualizados.

**8. Un sistema de ficheros UNIX utiliza bloques de 512 bytes y direcciones de disco de 16 bits. Los nodos-i contienen 10 direcciones de disco para bloques de datos, una dirección de bloque índice indirecto simple y una dirección de bloque índice indirecto doble. Conteste de manera razonada a las siguientes cuestiones:**

**a) ¿Cuál es el tamaño máximo de un fichero en este sistema?**

Analizamos 2 posibles limitaciones del tamaño del fichero (no tenemos información sobre el offset):

1. Número máximo de bloques:  

$$(10 + (2^9/2) + (2^9/2)^2) \text{ bloques} * 2^9 \text{ bytes/bloque} =$$

$$= 5*2^{10} + 2^{17} + 2^{23} \text{ bytes}$$
2. Tamaño de las direcciones del disco: con 16 bits para el direccionamiento de los bloques se pueden referenciar hasta  $2^{16}$  bloques, que equivalen a  

$$(2^{16} \text{ bloques}) * (2^9 \text{ bytes/bloque}) = 2^{25} \text{ bytes}$$

Lo que limita entonces el tamaño del fichero es el número máximo de bloques que podemos referenciar por nuestro sistema de nodos-i, es decir, podrá tener un tamaño máximo de  $5*2^{10} + 2^{17} + 2^{23}$  bytes = **8524800 bytes** (aprox,  $2^{23}$  bytes)



(a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

b) Un programa UNIX crea un fichero en este sistema e inmediatamente después escribe un byte de datos en la posición 1.000 y otro en la posición 10.000. ¿Cuántos bloques de datos ocupa este nuevo fichero en disco?

El byte 1.000 está en el BL1 (1.000/512), cuyo BF estará referenciado por el segundo puntero directo del nodo-i. Por otro lado, el byte 10.000 está en el BL19 (10.000/512), cuyo BF estará referenciado por el décimo puntero del BF apuntado por el puntero indirecto simple del nodo-i. Es decir, el fichero ocupará  $1 + (1 + 1) = 3$  bloques de disco.

9. Dado un sistema de ficheros tipo UNIX, en el que los directorios son relativamente pequeños (caben en un bloque de disco) y únicamente tiene una partición; razona qué operaciones de disco se necesitan para abrir el archivo "/usr/course3/SO/alumnos.txt". Suponer que el i-nodo del directorio raíz está ya en memoria y que no se ha cargado en memoria anteriormente ningún otro elemento de la ruta.

Para abrir el archivo necesitamos traer su nodo-i a la tabla de nodos-v en memoria. Por tanto, habrá que hacer lo siguiente:

Leer el nodo-i 2 (el del directorio raíz) en memoria para ver su bloque físico → Buscar y traer de disco ese bloque  
 → Leer el bloque hasta encontrar la entrada correspondiente al directorio 'course3' y ver su número de nodo-i → Buscar y traer de disco ese nodo-i → Leer el nodo-i para ver su bloque físico → Buscar y traer de disco ese bloque  
 → Leer el bloque hasta encontrar la entrada correspondiente al directorio 'SO' y ver su número de nodo-i → Buscar y traer de disco ese nodo-i → Leer el nodo-i para ver su bloque físico → Buscar y traer de disco ese bloque  
 → Leer el bloque hasta encontrar la entrada correspondiente al fichero 'alumnos.txt' y ver su número de nodo-i → Buscar y traer de disco ese nodo-i

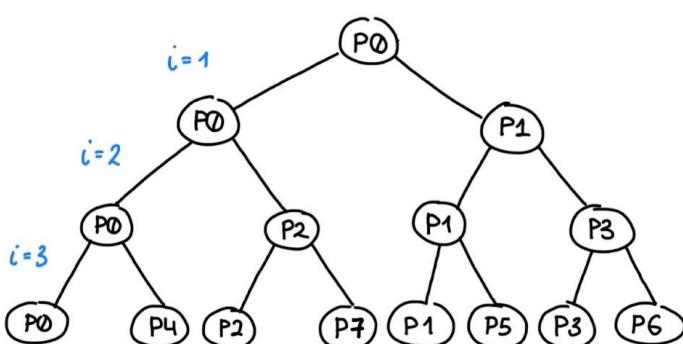
## PROCESOS

1. Dado el siguiente programa ejecutado bajo UNIX:

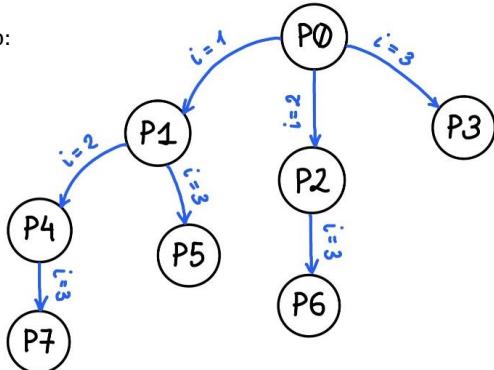
```
void main(int argc, char *argv[]) {
    int i;

    for (i=1; i<=argc; i++)
        fork();
    ...
```

Dibuje el esquema jerárquico de procesos que se genera para  $argc = 3$



Otra manera de representarlo:



## 2. Considere el siguiente código:

```
int varGlobal;

void main() {
    int varLocal=3;
    pid_t pid;

    varGlobal=10;
    printf("Soy el proceso original. Mi PID es %d\n", getpid());
    fflush(NULL);

    pid = fork();
    if (pid == -1) {
        perror("Error en fork()\n");
        exit(-1);
    }
    if (pid == 0 ) {
        // CODIGO DEL PROCESO HIJO
        varGlobal = varGlobal + 5;
        varLocal = varLocal + 5;
    }
    else {
        // CODIGO DEL PADRE: pid contiene el pid del hijo
        wait(NULL);
        varGlobal = varGlobal + 10;
        varLocal = varLocal + 10;
    }
    printf("Soy el proceso con PID %d. Mi padre es %d Global: %d Local %d\n",
    getpid(), getppid(),varGlobal, varLocal );
}
```

Asumiendo que el proceso original tiene PID 100 y es hijo del proceso *init* (PID=1), indica qué se mostrará por pantalla al ejecutar el código. ¿Es posible que los valores finales de las variables varíen de una ejecución a otra en función del orden de planificación? ¿Puede cambiar el orden en el que se muestran los diferentes mensajes por pantalla?

- Se mostrará por pantalla (asumiendo que los PID se asignan incrementando en 1, y que no se crea ningún proceso entre el padre y el hijo):  
Soy el proceso original. Mi PID es 100  
Soy el proceso con PID 101. Mi padre es 100 Global: 15 Local: 8  
Soy el proceso con PID 100. Mi padre es 1 Global: 20 Local: 13
- Los valores de las variables serán siempre los mismos, independientemente del orden en que se vayan ejecutando los procesos (que además en este caso siempre será el mismo, dado que el padre espera a que el hijo termine), porque las variables de cada proceso son independientes y no comparten el mapa de memoria.
- Si borramos el `wait(NULL)` del código del proceso padre, el orden en el que se mostrarán los mensajes será aleatorio, dependiendo del proceso que termine antes. Mientras esté presente, el mensaje del proceso hijo siempre aparecerá primero.

## 3. Considere el siguiente código:

```
int a = 3;
void main() {
    int b=2;
    for (i=0;i<4;i++) {
        p=fork();
        if (p==0) {
            b++;

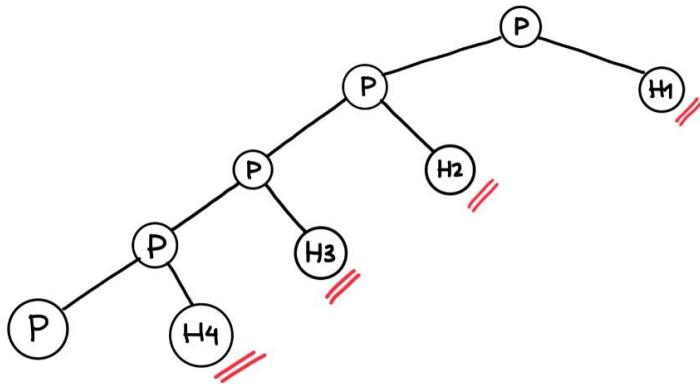
            execlp("comando":...);
            a++;
        }
    }
}
```

```

        else {
            wait();
            a++;
            b--;
        }
    }
    imprime(a,b);
}

```

a) ¿Cuántos procesos se crean en total? (sin contar el padre original) ¿Cuántos coexisten en el sistema como máximo?



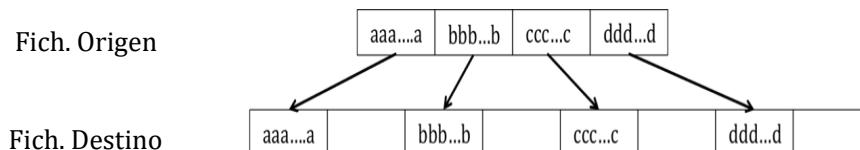
En principio, se crearán 4 procesos hijos, además del padre, dado que el bucle *for* se ejecuta 4 veces (en cada vuelta se crea un hijo), y los hijos no llegarán a generar más hijos, asumiendo que en el ejecutable “comando” no se haga otro *fork*, y asumiendo que se podrá ejecutar sin problemas (el archivo existe, es ejecutable, etc).

En el sistema coexistirán a lo sumo 2 procesos, el padre y el hijo que acabe de crear, dado que el padre espera a que el hijo muera antes de dar otra vuelta al bucle (y crear otro hijo).

b) ¿Qué se imprimirá al mostrar los valores de *a* y *b*?

Si todo va bien (el ejecutable “comando” existe, se puede ejecutar, etc), el único que llegará a la instrucción *imprime(a,b)* es el padre, ya que los hijos habrán cambiado su mapa de memoria al ejecutar *execvp()*. Por lo tanto, se imprimirá el valor de las variables *a* y *b* del padre (que son independientes de las de sus hijos), es decir, *a* = 7, *b* = -2 (*a++* y *b--* se ejecutan 4 veces).

**4.** Un programador poco avezado pretende hacer una aplicación que realice *copias intercaladas* de ficheros en paralelo. El concepto de copia intercalada se ilustra en la figura: se copia el primer bloque íntegro, y se deja un bloque vacío. Se copia el segundo bloque del fichero origen al tercer bloque del destino, y así sucesivamente.



El código de su programa para la *copia intercalada* de un fichero de 4 bloques mediante 4 procesos, es el siguiente:

```

#define BLOCK 4096
char buf[BLOCK] = "xxxxxxxx...xxxxxx";

void copia_bloque(int fdo, int fdd) {
    read(fdo,buf,BLOCK);
    write(fdd,buf,BLOCK);
}

void main() {
    pid_t pid;
    int fdo,fdd;
    fdo = open(«Origen»,O_RDONLY);

```

```

fdd = open(<<Destino>>, O_RDWR|O_CREAT|O_TRUNC, 0666);
for (int i=0; i < 4; i++) {
    lseek(fdo,i*BLOCK, SEEK_SET);
    lseek(fdd,2*i*BLOCK,SEEK_SET);
    pid = fork(fo);
    if (pid==0){
        copia_bloque(fdo,fdd);
        exit(0);
    }
}
while (wait(NULL)!=-1) { };
read(fdd,buf,BLOCK);
lseek(fdd,0,SEEK_SET);
read(fdd, buf,BLOCK);
}

```

Responde a las siguientes preguntas, suponiendo que la **prioridad es para el proceso padre**, y después para cada uno de los hijos en el orden de creación.

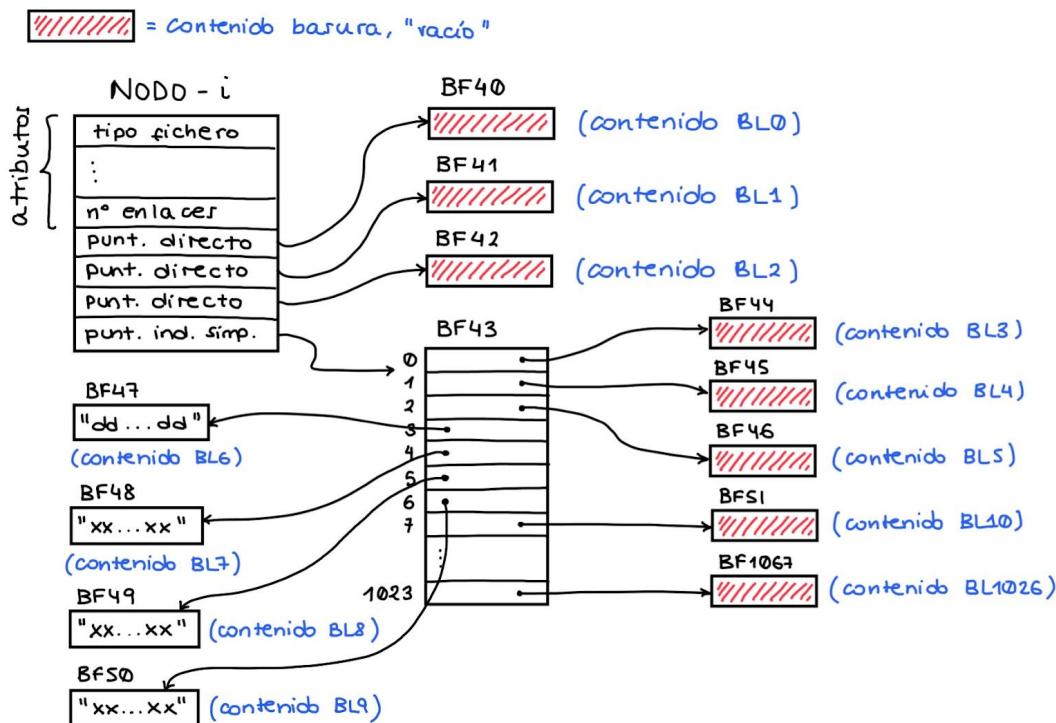
a) Indica el contenido del array **buf** inmediatamente después de la ejecución de las líneas 18 y 20. Justifica tu respuesta.

En primer lugar, el padre ejecuta el bucle 4 veces y, con él, los *lseek()*. Así, para cuando entra en el while y empieza a esperar a los hijos que ha generado (4), el puntero de L/E de la instancia del fichero ORIGEN apunta al byte 3x4096 (toca leer el bloque 3), y el del fichero DESTINO apunta al byte 6x4096 (toca escribir en el bloque 6).

Como el fichero es sólo de 4 bloques, el primer proceso hijo copiará el BL3 de ORIGEN al BL6 de DESTINO. El resto de hijos escribirán "xx...xx" en los bloques 7, 8 y 9 de DESTINO (no leerán nada de ORIGEN, por lo que su buf seguirá = "xx...xx").

Por lo tanto, cuando han hecho el *exit()* todos los hijos, en la línea 18 se intentará leer del fichero DESTINO, pero no podrá (el puntero de L/E apunta al final del fichero), por lo que buf contendrá "xx...xx" (su contenido inicial). En la línea 20 (después de haber puesto el puntero al principio del fichero) se leerá el BL0 de DESTINO, que estará "vacío" (con contenido basura).

b) Sea sistema de ficheros de tipo Linux (nodos-i), con 3 punteros directos y un indirecto simple, tamaño de bloque de 4KB (4096 bytes) y 4bytes por puntero. Dibuja un posible estado final de toda la información del sistema de ficheros relativa al fichero "Destino".





## (a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

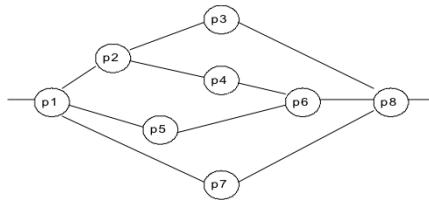
Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

(los números de los bloques físicos se han elegido aleatoriamente)

5. Use las llamadas *fork()*, *exec()*, *exit()* y *wait()* de UNIX para describir la sincronización de los ocho procesos cuyo grafo general de precedencia es el siguiente. Para ello escriba un función *main()* en la que se vayan creando los 8 procesos mediante llamadas a *fork()*, se ejecuten los binarios asociados a cada proceso (por ejemplo, *p1.out* para el proceso *p1*...) y se respete la precedencia mostrada en la figura (por ejemplo, *p6* no puede comenzar hasta que no hayan acabado *p4* y *p5*).



```

void main() {

    pid_t pidP1, pidP2, pidP3, pidP4, pidP5, pidP6, pidP7, pidP8;

    pidP1 = fork();
    if (pidP1 == 0)
        execl("P1.out", NULL);
    wait(NULL);
    pidP2 = fork();
    if (pidP2 == 0)
        execl("P2.out", NULL);
    pidP5 = fork();
    if (pidP5 == 0)
        execl("P5.out", NULL);
    pidP7 = fork();
    if (pidP7 == 0)
        execl("P7.out", NULL);
    waitpid(pidP2, NULL);
    pidP3 = fork();
    if (pidP3 == 0)
        execl("P3.out", NULL);
    pidP4 = fork();
    if (pidP4 == 0)
        execl("P4.out", NULL);
    waitpid(pidP4, NULL);
    waitpid(pidP5, NULL);
    pidP6 = fork();
    if (pidP6 == 0)
        execl("P6.out", NULL);
    wait(NULL);
    pidP8 = fork();
    if (pidP8 == 0)
        execl("P8.out", NULL);

}
  
```

6. Considere el siguiente código:

```

int fd = -1;
char buf1[4] = "aaaa";
char buf2[4] = "bbbb";
int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, h1, NULL);
    pthread_create(&tid2, NULL, h2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
  
```

```

        close(fd);
    }
void* h1( ) {
    fd=open ("prueba",O_RDWR|O_CREAT|O_TRUNC,0666);
    write(fd,buf1,4);
}
void* h2( ) {
    while (fd== -1) {};
    write(fd,buf2,4);
}

```

Indique si cada una de las siguientes afirmaciones es cierta o falsa justificando la respuesta:

a) La escritura del hilo 2 (función h2) producirá un error por no haber abierto antes el fichero.

FALSO: Para cuando el hilo 2 ejecute el `write()`, habrá salido del bucle siendo `fd` distinto de -1, por lo que el hilo 1 ya habrá abierto el fichero. Dado que la TDDA y la tabla intermedia son compartidas para ambos hilos, la apertura del archivo será también visible para el hilo 2, y podrá escribir sin problema (asumiendo que el `open()` se haya realizado con éxito).

b) El contenido final del fichero prueba será o bien, “aaaa” o bien “bbbb”; ninguna otra alternativa es posible.

FALSO: El contenido del fichero será “aaaabbbb”, o bien “bbbbaaaa”, dependiendo de qué hilo realice primero la escritura. Si no se produce ninguna excepción, siempre escribirá primero un hilo (actualizando el puntero de L/E) y a continuación el otro, por lo que en ningún caso el contenido del fichero sería sólo “aaaa” o “bbbb”.

c) La llamada a `close()` del programa principal no debería devolver ‘-1’ (esto es, no debería producir ningún error).

VERDADERO: `close()` podría producir un error si se llamase a la función antes de abrir el fichero (no hay ninguna entrada que eliminar). Sin embargo, el hilo principal espera mediante `pthread_join()` a que el hilo 1, que es quien hace el `open()`, termine. En principio, por tanto, no debería producir ningún error, asumiendo que el `open()` acabe con éxito.

7. En un sistema monoprocesador los siguientes trabajos llegan a procesarse en los instantes indicados. ¿Cuáles son los tiempos de retorno (turnaround) y de espera para cada uno de ellos, los tiempos de retorno y de espera promedios, así como la productividad (throughput) del sistema, aplicando las diferentes estrategias de planificación listadas? Aplique los algoritmos de planificación sin expropiación y base las decisiones en la información de que se dispone en el momento de tomarlas. El desglose de los tiempos de los trabajos se refiere al tiempo requerido para cadenas de uso de UCP y de E/S alternativamente.

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

a) SPN: Primero el de menor tiempo de UCP siguiente

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
T1	X	O	O	O	O	O	-	-	X																				
T2		X	X	X	O	-	-	X	O																				
T3	-	-	-	-	-	-	-	-	-	X	X	X	X	X	O	O	O	O	X										
T4			-	X	X	X	O	O	X	O																			
Colas																													
C0		T3	T3	T3	T4	T3	T2	T2	T1	T3	T3																		

	T1	T2	T3	T4	Media
T Espera	2	2	10	1	3.7
T Retorno	9	8	20	8	11.1
Productividad	$4/20 = 0.2 \text{ trab}$				
Uso de CPU (%)	80.00%				

b) RR: Por turnos con quanto 2

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
T1	x	o	o	o	o	o	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
T2	-	-	-	x	x	-	-	-	-	x	o	-	-	x	o	-	-	-	-	-	-	-	-	-	-	-	-		
T3	-	x	x	-	-	-	x	x	-	-	-	x	o	o	o	o	x	-	-	-	-	-	-	-	-	-	-		
T4	-	-	-	x	x	-	-	-	-	x	o	o	x	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Colas																													
C0	T3	T2	T2	T4	T4	T3	T3	T2	T2	T1	T4	T3	T2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	T3	T3	T2	T2	T1	T1	T4	T3	T2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	T1	T4	T4	T3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

	T1	T2	T3	T4	Media
T Espera	4	8	8	6	5.1
T Retorno	11	14	18	13	14
Productividad	$4/18 = 0.22 \text{ tra}$				
Uso de CPU (%)	88.89%				

8. Cinco trabajos de lotes, de A a E, llegan casi al mismo tiempo a un centro de cálculo dotado de un único procesador. La estimación de sus respectivos tiempos de ejecución es de 10, 6, 2, 4, y 8 minutos. Sus prioridades, determinadas externamente, son 3, 5, 2, 1 y 4, respectivamente, siendo 5 la prioridad superior. Para cada uno de los siguientes algoritmos de planificación determine el tiempo medio de retorno de los procesos. (Ignorar el tiempo de comutación de procesos).

a) Por prioridad estricta.

Como se trata de un algoritmo no expropiativo, cuando un proceso empieza a usar la CPU no la abandonará hasta finalizar su ejecución. El orden de ejecución (toma de la CPU) de los procesos será B,E,A,C,D, y sus tiempos de retorno serán:

- B: 6 ud tiempo
- E:  $6 + 8 = 14$  ud tiempo
- A:  $6 + 8 + 10 = 24$  ud tiempo
- C:  $6 + 8 + 10 + 2 = 26$  ud tiempo
- D:  $6 + 8 + 10 + 2 + 4 = 30$  ud tiempo

$$\Rightarrow \text{Tiempo de retorno medio} = (6+14+24+26+30)/5 = 20 \text{ ud tiempo}$$

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
T1 - 10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
T2 - 6	x	x	x	x	x	x																								
T3 - 2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x					
T4 - 4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x		
T5 - 8	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x															
Colas	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T3	T4	T4														
C0	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T4																
	T4	T4	T4	T4	T4	T4	T4	T4	T4	T4	T4	T4	T4																	
	T5	T5	T5	T5	T5	T5	T5																							
	T1	T2	T3	T4	T5	Media																								
	T\_espera	14	0	24	26	6	13.																							
	T\_retorno	24	6	26	30	14	20																							
	Productividad	$5/30 = 0,17 \text{ tra}$																												
	Uso de CPU (%)	100.00%																												

### b) Por orden de llegada (FCFS)

El enunciado indica que llegan casi al mismo tiempo, por lo que no podemos saber exactamente el orden en el que llegan al sistema los procesos, pero habrá alguno. Suponiendo que llegasen en orden A-B-C-D-E, y teniendo en cuenta que se trata de un algoritmo no expropiativo, los tiempos de retorno serían:

- A: 10 ud tiempo
- B:  $10 + 6 = 16$  ud tiempo
- C:  $10 + 6 + 2 = 18$  ud tiempo
- D:  $10 + 6 + 2 + 4 = 22$  ud tiempo
- E:  $10 + 6 + 2 + 4 + 8 = 30$  ud tiempo

=> Tiempo de retorno medio =  $(10+16+18+22+30)/5 = 19,2$  ud tiempo

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
T1 - 10	x	x	x	x	x	x	x	x	x	x																				
T2 - 6	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x														
T3 - 2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x														
T4 - 4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x									
T5 - 8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x		
Colas	T2	T2	T2	T2	T2	T2	T2	T2	T2	T3	T3	T3	T3	T3	T3	T4	T4	T5	T5	T5	T5									
C0	T3	T3	T3	T3	T3	T3	T3	T3	T4	T4	T4	T4	T4	T5																
	T4	T4	T4	T4	T4	T4	T4	T4	T4	T4	T5																			
	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5																				
	T1	T2	T3	T4	T5	Media																								
	T\_espera	0	10	16	18	22	12.																							
	T\_retorno	10	16	18	22	30	19.																							
	Productividad	$5/30 = 0,17 \text{ tra}$																												
	Uso de CPU (%)	100.00%																												



(a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

c) Por menor tiempo de ejecución (SPN)

Como se trata de un algoritmo no expropiativo, cuando un proceso empieza a usar la CPU no la abandonará hasta finalizar su ejecución. El orden de ejecución (toma de la CPU) de los procesos será C,D,B,E,A, y sus tiempos de retorno serán:

- C: 2 ud tiempo
- D:  $2 + 4 = 6$  ud tiempo
- B:  $2 + 4 + 6 = 12$  ud tiempo
- E:  $2 + 4 + 6 + 8 = 20$  ud tiempo
- A:  $2 + 4 + 6 + 8 + 10 = 30$  ud tiempo

=> Tiempo de retorno medio =  $(2+6+12+20+30)/5 = 14$  ud tiempo

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29			
T1 - 10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x					
T2 - 6	-	-	-	-	-	-	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-					
T3 - 2	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-					
T4 - 4	-	-	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-					
T5 - 8	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-					
Colas																																	
C0	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1			
	T2	T2	T2	T2	T2	T2	T5																										
	T4	T4	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5	T5		
	T5	T5																															
	T1	T2	T3	T4	T5	Media																											
T Espera	20	6	0	2	12	7.1																											
T Retorno	30	12	2	6	20	14																											
Productividad	5/30 = 0,17 tra																																
Uso de CPU (%)	100.00%																																

9. Considere un sistema monoprocesador con una política de planificación de procesos de 3 niveles con realimentación. Cada nivel usa a su vez una política de planificación circular (round robin) cuyos cuantos de tiempo son 2, 4 y 8, respectivamente. Al principio hay 3 procesos en la cola del nivel 1 (máxima prioridad). Los patrones de ejecución de los procesos son los siguientes (y se repiten indefinidamente):

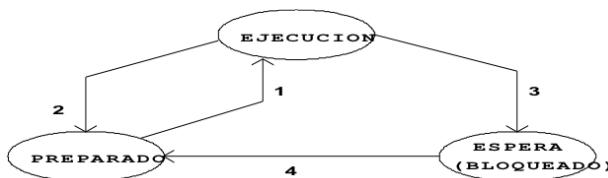
P1 (3-CPU,5-E/S)    P2 (8-CPU,5-E/S)    P3 (5-CPU,5-E/S)

Las colas de los otros dos niveles están vacías. Cuando acaba una operación de E/S, los procesos entran en la cola de mayor prioridad. Usando un diagrama de tiempos muestre qué proceso está ejecutándose y qué procesos hay en cada nivel, durante las 30 primeras unidades de tiempo de ejecución. Calcule además la utilización de CPU y los tiempos de espera de cada proceso.

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
T1	X	X					X	o	o	o	o	o	X	X		X	o	o	o	o	o	X	X		X	X				
T2			X	X				X	X	X	X						X	X	o	o	o	o	o	X	X					
T3				X	X						X					X	X	o	o	o	o	X	X			X	X	X		
Colas																														
C0 RR q=2		T2	T2	T3	T3								T1	T1									T1	T2						
C1 RR q=4		T1	T1	T1	T1	T2	T3	T1	T1	T1	T1	T1																		
C2 RR q=8																									T2	T2	T2	T2	T2	

Utilización CPU	93.
Tiempo de espera de T1	12
Tiempo de espera de T2	15
Tiempo de espera de T3	15

Podemos describir gran parte de la gestión del procesador en términos de diagramas de transiciones de estado como éste:



a) ¿Qué "evento" causa cada una de las transiciones marcadas?

- 1 → El planificador decide dar la CPU a un proceso que estaba esperando (porque es más prioritario, es su turno, etc)
- 2 → El planificador decide quitar la CPU a un proceso que estaba en ejecución (porque se ha acabado su quanto, hay otro más prioritario, etc)
- 3 → Un proceso se bloquea porque, por ejemplo, está esperando un dato de E/S
- 4 → El proceso ya ha recibido el dato de E/S y pasa a estar listo para seguir ejecutándose

b) Si consideramos todos los procesos del sistema, podemos observar que una transición de estado por parte de un proceso podría hacer que otro proceso efectuara una transición también. ¿Bajo qué circunstancias podría la transición 3 de un proceso provocar la transición 1 inmediata de otro proceso?

En general, cuando un proceso se bloquee (transición 3), el planificador pondrá otro proceso en ejecución (transición 1). Es decir, siempre que haya algún proceso esperando, cuando otro se bloquee este tomará el control de la CPU.

c) ¿Bajo qué circunstancias, si las hay, podrían ocurrir las siguientes transiciones causa-efecto?

i. 2 → 1 (por el hecho de que ocurra una transición tipo 2, hay una transición 1)

Un proceso agota su quanto, pasa a estar por tanto listo para ejecutarse, y otro proceso (incluso él mismo, si no hubiera más o le tocase a él según la política) entrará en ejecución.

ii. 3 → 2

No es posible. Si un proceso pasa a bloquearse no puede pasar también a estar listo para ejecutarse (ya que no hay más de 1 proceso en ejecución). Si la espera es inmediata, podría incluso darse 3→4, pero en ningún caso es causa efecto 3→2.

iii. 4 → 1

Un proceso termina su espera (recibe el dato de E/S, etc) y no hay ningún hilo preparado para ejecutarse, por lo que toma él el control de la CPU.

d) ¿Bajo qué circunstancias, si las hay, las transiciones 1, 2, 3 y 4 NO producirían ninguna otra transición inmediata?

- Si un proceso se bloquea (transición 3) y no hay ningún proceso esperando, no habría ninguna transición inmediata.
- Si un proceso termina de estar bloqueado (transición 4), en general no habrá ninguna transición inmediata, a no ser que se de el caso del apartado c.iii.
- Si un proceso pasa a estar en ejecución, no tiene por qué producirse ninguna transición inmediata, no desencadena nada.

**11.** Considere un sistema monoprocesador con una planificación MLF (multinivel con realimentación) donde el número de niveles es  $n = 10$ , y el intervalo de planificación para el nivel  $i$  es  $T_i = 2i \cdot q$ , siendo  $q$  es el valor del intervalo básico de planificación o quanto. En nuestro sistema sólo hay tres procesos, se encuentran inicialmente en la cola de máxima prioridad T1 y sus tiempos respectivos hasta la siguiente petición de E/S son 3, 8, y 5 quantos. Cuando un proceso alcanza su petición de E/S permanece suspendido (sin competir por la CPU) durante 5 unidades de tiempo, tras las cuales vuelve a entrar en la cola de máxima prioridad. Los tiempos de CPU requeridos hasta la petición de E/S siguiente son de nuevo 3, 8, y 5. Usando un diagrama de tiempos, muestre qué proceso estará ejecutándose y qué procesos estarán en qué cola durante cada una de las primeras 30 unidades de tiempo de ejecución, y cuáles son los valores de las siguientes medidas de rendimiento: productividad, tiempos retorno (turnaround) individual y promedio, y tiempos de espera individual y promedio.

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
T1	x	x				x	o	o	o	o	o	x	x	x	o	o	o	o	o	o	o	x	x	x	x							
T2			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x				
T3		x	x				x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
Colas																																
C1	T2	T2	T3	T3																												
	T3	T3																														
C2	T1	T1	T1	T1	T2	T3																										
	T2	T2	T3																													
C3																																

2q	T_espresa	T1	T2	T3	Media
	T_retorno	...	...	...	...
4q	Productividad	...			
	Uso de CPU (%)	93.33%			
8q					

**12.** Repita el ejercicio anterior, pero suponiendo que  $T_i = 2 \cdot q$  (es decir, constante), para todos los niveles de prioridad.

Tarea	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29			
T1	x	x			x	o	o	o	o	x	x	x	x	o	o	o	o	x	x	x	x	x	x	x	x	x	x	x	x				
T2		x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x				
T3		x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x				
Colas																																	
C1	T2	T2	T3	T3																													
	T3	T3																															
C2	T1	T1	T1	T1	T2	T3																											
	T2	T2	T3																														
C3																																	

2q	T_espresa	T1	T2	T3	Media
	T_retorno	...	...	...	...
2q	Productividad	...			
	Uso de CPU (%)	96.67%			
2q					

## SINCRONIZACIÓN

**1.** Escriba un programa que cree tres hilos que se comunicarán entre ellos. El hilo 1 genera los 1000 primeros números pares y el hilo 2 los 1000 números impares. El hilo 3 irá leyendo esos números e imprimiéndolos en pantalla. Se debe garantizar que los números escritos por pantalla estén en orden: 1,2,3,4,5....Implementar dicha sincronización mediante:

a) Cerrojos y variables condicionales.

```
#DEFINE N 2000
int n;
typedef enum {
    PAR,
    IMPAR,
```

```

    CONS
} type_turno;
type_turno turno;
pthread_cond_t c_par, c_imp, c_cons;
pthread_mutex mut;
int done_imp = 0, done_par = 0;

void* impares (void* arg) {
    for (int i = 1; i <= N; i+=2) {
        lock(mut);
        while (turno != IMPAR)
            cond_wait(c_imp, mut);
        n = i;
        turno = CONS;
        signal(c_cons);
        unlock(mut);
    }
    done_imp = 1;
}

void* pares (void* arg) {
    for (int i = 2; i <= N; i+=2) {
        lock(mut);
        while (turno != PAR)
            cond_wait(c_par, mut);
        n = i;
        turno = CONS;
        signal(c_cons);
        unlock(mut);
    }
    done_par = 1;
}

void* consumer(void* arg) {
    while(!done_imp || !done_par) {
        lock(mut);
        while (turno != CONS)
            cond_wait(c_cons, mut);
        printf(out, &d\n, n);
        if (n % 2 == 0) {
            turno = IMPAR;
            signal(c_impar);
        }
        else {
            turno = PAR;
            signal(c_par);
        }
        unlock(mut);
    }
}

int main() {
    turno = IMPAR;
    pthread_t tidImp, tidPar, tidCons;

    // creamos el mutex y las variables condicionales
    init(&mut);
    init(&c_imp);
    init(&c_par);
    init(&c_cons);
}

```

# Que no te escriban poemas de amor cuando terminen la carrera ➤➤➤➤➤



WUOLAH

(a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

```
// creamos hilos
pthread_create(tidImp, NULL, impares, NULL);
pthread_create(tidPar, NULL, pares, NULL);
pthread_create(tidCons, NULL, consumer, NULL);

pthread_join(tidImp);
pthread_join(tidPar);
pthread_join(cons);

destroy(&mut);
destroy(&c_imp);
destroy(&c_par);
destroy(&c_cons);

exit(0);

}
```

## 2. (otros)

```
void hiloTemp( ) {
    while (1) {
        float t = leerSensorTemp();

        /* 1. Dormir si hay 2 muestras de Temp
         * pendientes de envío.
         * 2. escribir dato t en medidas
         * 3. despertar a hiloEnvio si procede
        */

        pthread_mutex_lock(&m);
        while (tempo == 2)
            pthread_cond_wait(&sensores, &m);
        if (medidas[0] == -1)
            medidas[0] = t;
        else
            medidas[1] = t;
        tempo++;
        if (tempo == 2)
            pthread_cond_signal(&envio);
        pthread_mutex_unlock(&m);

        // dormir hasta siguiente muestra
        sleep(n);
    }
}

void hiloHumedad( ) {
    while (1) {
        float h = leerSensorHumedad();
        pthread_mutex_lock(&m);
        while (humo == 2)
            pthread_cond_wait(&sensores, &m);
        if (medidas[2] == -1)
            medidas[2] = h;
        else
            medidas[3] = h;
        humo++;
        if (humo == 2)
            pthread_cond_signal(&envio);
        pthread_mutex_unlock(&m);

        sleep(n);
    }
}
```

```
// declaración de variables
float medidas[4] = {-1,-1,-1,-1};
int tempo = 0, humo = 0;
mutex_t m;
cond_t sensores, envio;

void hiloEnvio() {
    /* 1. Esperar a que deba enviar
     * 2. Enviar todas las muestras en medidas
     * (usando enviaMuestras() tantas veces como sea
     * necesario)
     * 3. despertar a otros hilos si procede
    */
    while (1) {
        pthread_mutex_lock(&m);
        while (tempo != 2 || humo != 2)
            pthread_cond_wait(&envio, &m);
        for (int i = 0; i < 4; ++i)
            enviaMuestras(medidas[i]);
        medidas[i] = -1;
    }
    tempo = 0;
    humo = 0;
    pthread_cond_broadcast(&sensores);
    pthread_mutex_unlock(&m);
}
```

### 3. (otros)

<pre> typedef struct {     sem_t s;     mutex_t m;     /* Completar si es necesario */     int n; //cuántos están esperando } varcond_t; </pre>	<pre> int varcond_init(varcond_t *vc) {     /* COMPLETAR */     sem_init(&amp;vc-&gt;s, ..., 0);     pthread_mutex_init(&amp;vc-&gt;m);     vc-&gt;n = 0; }  int varcond_wait(varcond_t *vc, mutex_t* mtx) {     /* COMPLETAR */     pthread_mutex_unlock(mtx);     //actualizamos n     pthread_mutex_lock(&amp;vc-&gt;m);     ++n;     pthread_mutex_unlock(&amp;vc-&gt;m);     //bloqueamos hilo     sem_wait(&amp;vc-&gt;s);     //actualizamos n     pthread_mutex_lock(&amp;vc-&gt;m);     --n;     pthread_mutex_unlock(&amp;vc-&gt;m);     pthread_mutex_lock(mtx); } </pre>
<pre> int varcond_signal(varcond_t* vc) {     /* COMPLETAR */     pthread_mutex_lock(&amp;vc-&gt;m);     if(n &gt; 0)         sem_post(&amp;vc-&gt;s);     pthread_mutex_unlock(&amp;vc-&gt;m); }  int varcond_broadcast(varcond_t* vc) {     /* COMPLETAR */     pthread_mutex_lock(&amp;vc-&gt;m);     for(int i = 0; i &lt; n; ++i)         sem_post(&amp;vc-&gt;s);     pthread_mutex_unlock(&amp;vc-&gt;m); } </pre>	

## MEMORIA

### 1. Considerar los cuatro sistemas siguientes:

	A	B	C	D
Tamaño de página (en palabras)	512	512	1024	1024
Tamaño de palabra (en bits)	16	32	16	32
SOL	64	$2^{21}$		

Para cada sistema determinar el número de entradas de la tabla de páginas (TMP). Suponer que sólo existe una TMP para cada sistema y que cada dirección virtual ocupa una palabra (16 o 32b).

Para todos los apartados, el nº de entradas de la TP es el número de páginas virtuales, por lo que basta dividir el tamaño de la memoria (todas las direcciones posibles) entre el tamaño de una página para saber el número de páginas.

- A →  $2^{16} / 2^9 = 2^7 = 128$  entradas
- B →  $2^{32} / 2^9 = 2^{23}$  entradas
- C →  $2^{16} / 2^{10} = 2^6 = 64$  entradas
- D →  $2^{32} / 2^{10} = 2^{22}$  entradas

2. En un sistema de paginación por demanda se obtiene que, con cierta carga de trabajo, la CPU emplea un 15 % del tiempo y el disco de swap está ocupado el 92 % del tiempo. ¿Qué acción aumentaría más la utilización de la CPU?

El hecho de que el disco de swap esté tan ocupado se deberá a que habrá continuamente fallos de página que obligarán a sustituir todo el tiempo páginas de la memoria principal, para lo que habrá que ir trayendo las nuevas páginas de disco. Por tanto, lo que aumentará más la utilización de la CPU o, dicho de otra manera, lo que reducirá el tiempo del disco ocupado es ampliar la memoria principal, para incrementar el número de páginas que pueda almacenar, y reducir así el número de fallos de página.

3. Si se utilizase sustitución de páginas FIFO con cuatro marcos de página y ocho páginas, ¿cuántos fallos de página ocurrirían para la cadena de referencias 0 1 7 2 3 2 7 1 0 3 si los cuatro marcos estuvieran inicialmente vacíos? Repetir el problema para LRU.

FIFO => 6 fallos de página

	0	1	7	2	3	2	7	1	0	3
MP0	0*	0	0	0	3*	3	3	3	3	3
MP1	-	1*	1	1	1	1	1	1	0*	0
MP2	-	-	7*	7	7	7	7	7	7	7
MP3	-	-	-	2*	2	2	2	2	2	2

LRU => 7 fallos de página

	0	1	7	2	3	2	7	1	0	3
MP0	0*	0	0	0	3*	3	3	3	0*	0
MP1	-	1*	1	1	1	1	1	1	1	1
MP2	-	-	7*	7	7	7	7	7	7	7
MP3	-	-	-	2*	2	2	2	2	2	3*

4. Suponiendo una memoria física de cuatro marcos de página, indicar el número de fallos de página que genera la cadena de referencias a b g a d e a b a d e g d e para cada una de las siguientes políticas de sustitución. (Inicialmente todos los marcos están vacíos): óptima, FIFO, reloj y LRU.

OPT => 6 fallos de página

	a	b	g	a	d	e	a	b	a	d	e	g	d	e
MP0	a*	a	a	a	a	a	a	a	a	a	a	g*	g	g
MP1	-	b*	b	b	b	b	b	b	b	b	b	b	b	b
MP2	-	-	g*	g	g	e*	e	e	e	e	e	e	e	e
MP3	-	-	-	-	d*	d	d	d	d	d	d	d	d	d

FIFO => 10 fallos de página

	a	b	g	a	d	e	a	b	a	d	e	g	d	e
MP0	a*	a	a	a	a	e*	e	e	e	e	e	d*	d	
MP1	-	b*	b	b	b	b	a*	a	a	a	a	a	a	e*
MP2	-	-	g*	g	g	g	g	b*	b	b	b	b	b	b
MP3	-	-	-	-	d*	d	d	d	d	d	d	g*	g	g

RELOJ => 7 fallos de página

	a	b	g	a	d	e	a	b	a	d	e	g	d	e
MP0	↓ a*	↓ a	↓ a	↓ a .	↓ a .	a	a .	a .	a .	a .	a .	a	a	a
MP1	-	b*	b	b	b	e*	e	e	e	e	e	e	e	e
MP2	-	-	g*	g	g	↓ g	↓ g	b*	b	b	b	g*	g	g
MP3	-	-	-	-	d*	d	d	↓ d	↓ d	↓ d	↓ d	↓ d	↓ d	↓ d

LRU => 7 fallos de página

	a	b	g	a	d	e	a	b	a	d	e	g	d	e
MP0	a*	a	a	a	a	a	a	a	a	a	a	a	a	a
MP1	-	b*	b	b	b	e*	e	e	e	e	e	e	e	e
MP2	-	-	g*	g	g	g	g	b*	b	b	b	g*	g	g
MP3	-	-	-	-	d*	d	d	d	d	d	d	d	d	d

5. Considerar la siguiente secuencia de referencias a memoria virtual generadas por un sólo programa en un sistema con paginación pura: 10, 11, 104, 170, 73, 309, 185, 245, 246, 434, 458, 364.

a) Deducir la correspondiente cadena de referencias, suponiendo un tamaño de página de 100 palabras 0,0,1,1,0,3,1,2,2,4,4,3 (basta tomar la parte entera de la división de la referencia entre 100)

b) Determinar el número de fallos de página para cada una de las siguientes estrategias de sustitución, suponiendo que hay dos marcos de página disponibles para el programa: OPT, FIFO, Reloj y LRU.

OPT => 5 fallos de página

	10	11	104	170	73	309	185	245	246	434	458	364
MP0	0*	0	0	0	0	3*	3	3	3	3	3	3
MP1	-	-	1*	1	1	1	1	2*	2	4*	4	4

FIFO => 6 fallos de página

	10	11	104	170	73	309	185	245	246	434	458	364
MP0	0*	0	0	0	0	3*	3	3	3	4*	4	4
MP1	-	-	1*	1	1	1	1	2*	2	2	2	3*

RELOJ => 6 fallos de página

	10	11	104	170	73	309	185	245	246	434	458	364
MP0	↓ 0*	↓ 0 .	↓ 0 .	↓ 0 .	↓ 0 .	3*	3	2*	2	↓ 2	↓ 2	3*
MP1	-	-	1*	1 .	1 .	↓ 1	↓ 1 .	↓ 1	↓ 1	4*	4 .	↓ 4 .

LRU => 7 fallos de página

	10	11	104	170	73	309	185	245	246	434	458	364
MP0	0*	0	0	0	0	0	1*	1	1	4*	4	4
MP1	-	-	1*	1 .	1 .	3*	3	2*	2	2	2	3*

6. Considera un sistema de paginación bajo demanda en el que un proceso que tiene asignados 3 marcos de página genera la siguiente secuencia direcciones virtuales:

0x2F, 0x30, 0x11, 0x2A, 0x41, 0x5F, 0x2A, 0x30, 0x10, 0x5A, 0x6B, 0x11

Que no te escriban poemas de amor  
cuando terminen la carrera ➤➤➤➤➤



WUOLAH

(a nosotros por suerte nos pasa)

No si antes decíte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

- b) Indica qué accesos producirían un fallo de página cuando se utilizan políticas de reemplazo local FIFO y LRU.

FIFO => 10 fallos de página

	0x2F	0x30	0x11	0x2A	0x41	0x5F	0x2A	0x30	0x10	0x5A	0x6B	0x11
MP0	2*	2	2	2	4*	4	4	3*	3	3	6*	6
MP1	-	3*	3	3	3	5*	5	5	1*	1	1	1
MP2	-	-	1*	1	1	1	2*	2	2	5*	5	5

LRU => 9 fallos de página

	0x2F	0x30	0x11	0x2A	0x41	0x5F	0x2A	0x30	0x10	0x5A	0x6B	0x11
MP0	2*	2	2	2	2	2	2	2	2	5*	5	5
MP1	-	3*	3	3	4*	4	4	3*	3	3	6*	6
MP2	-	-	1*	1	1	5*	5	5	1*	1	1	1

- c) ¿Sería beneficioso aumentar el número de marcos de página asignados al proceso hasta 4 para alguna de estas dos políticas? Justifica tu respuesta indicando el número de fallos de página que se producirían en esta nueva situación para cada algoritmo.

FIFO => 9 fallos de página

(reduce sólo 1 fallo, no  
se hasta qué punto compensa...)

	0x2F	0x30	0x11	0x2A	0x41	0x5F	0x2A	0x30	0x10	0x5A	0x6B	0x11
MP0	2*	2	2	2	2	5*	5	5	5	5	6*	6
MP1	-	3*	3	3	3	3	2*	2	2	2	2	2
MP2	-	-	1*	1	1	1	1	3*	3	3	3	3
MP3	-	-	-	-	4*	4	4	4	1*	1	1	1

LRU => 8 fallos de página

(reduce sólo 1 fallo, no  
se hasta qué punto compensa...)

	0x2F	0x30	0x11	0x2A	0x41	0x5F	0x2A	0x30	0x10	0x5A	0x6B	0x11
MP0	2*	2	2	2	2	2	2	2	2	2	6*	6
MP1	-	3*	3	3	3	5*	5	5	5	5	5	5
MP2	-	-	1*	1	1	1	1	3*	3	3	3	3
MP3	-	-	-	-	4*	4	4	4	1*	1	1	1