Ejercicio 1
Se plantean tres cuestiones independientes sobre el sistema de ficheros de UNIX:

a) Para realizar su labor, el código del sistema de ficheros debe llevar a cabo numerosos cálculos aplicando diversas fórmulas. Suponiendo los siguientes parámetros genéricos:

- **TB**: tamaño del bloque en bytes; **TI**: tamaño del inodo en bytes (divisor exacto de **TB**).
- Inodo con **D** punteros directos, un indirecto simple y uno doble (por simplicidad, no hay indirecto triple). Un puntero a un bloque ocupa **P** bytes.
- El mapa de bloques libres está almacenado a partir del bloque **MB** (recuerde que este mapa sólo guarda el estado de bloques de datos) y el de inodos libres a partir de **MI** (recuerde que no existe el inodo 0).
- Los inodos están almacenados en el disco a partir del bloque **BI** y los datos a partir del bloque **BD**.

A partir de dichos parámetros, se pide especificar las siguientes fórmulas (**NOTA**: sólo se tendrá en cuenta una respuesta si incluye la fórmula solicitada, no siendo considerada si aparece únicamente una descripción textual):

a.1) Dado un número de inodo (**I**), desarrolle las fórmulas que calculan en qué bloque de disco (**B**) se encuentra y a partir de qué byte (**b**) dentro de dicho bloque se almacena el inodo.
a.2) Dado un número de bloque de datos (**X**), calcule en qué bloque del disco (**B**) se encuentra su información de estado (libre/ocupado) y, además, de qué bit (**b**) dentro de ese bloque se trata.
a.3) Igual que el apartado anterior pero siendo **X** el número de un inodo.
a.4) Dado un determinado byte (**b**) de un fichero, calcule, en primer lugar, a qué número de bloque (**BF**) del fichero pertenece. A continuación, especifique la fórmula que, dado un número de bloque dentro del fichero (**BF**) y su inodo asociado (**I**), determina en qué bloque de disco (**BD**) está almacenado. Esta fórmula se descompondrá en varias subfórmulas dependiendo de a qué intervalo pertenezca el bloque del fichero. Se deben especificar los rangos de los distintos intervalos basándose en los parámetros del sistema de ficheros y detallar la subfórmula aplicable a cada uno. Para ello, se utilizará la siguiente notación:

- **I[X]**: Corresponde con el puntero a bloque almacenado en la posición **X** del vector de punteros incluido en el inodo.
- **B(X)**: Bloque **X** del disco.
- **B(X)[Y]**: Dentro del bloque **X** del disco (que será un bloque de índices), corresponde con el puntero almacenado en la posición **Y**.

¿Encuentra alguna similitud entre este proceso de traducción y el usado en alguna otra parte del sistema operativo?
a.5) Dada la petición general `read(d,buf,tam)` y tomando como base las fórmulas anteriores, explique cómo se determina, a partir de los parámetros de la llamada, la lista de bloques del fichero involucrados y, a continuación, la lista de bloques de discos implicados. Se deberá explicar qué estructuras en memoria del sistema operativo se consultan, cómo se tiene acceso a ellas y qué datos se extraen de las mismas.

b) Supóngase que el tamaño del bloque es de 4096 bytes, que el número de enlaces directos es 12 y que existe un fichero de 81.940 (4096 * 20 + 20) bytes. Se plantean dos formas de leer completamente el fichero:

1) 20 lecturas de 4096 bytes y una final de 20 bytes.
2) 1 primera lectura de 20 bytes y 20 de 4096 bytes.

Suponiendo que en la cache de bloques no hay ninguna información sobre ese fichero, calcule para cada caso cuántos accesos a la cache del sistema de ficheros se producen y cuál es el porcentaje de aciertos, así como cuántos accesos al disco se realizan. Compare la eficiencia de estas estrategias.

c) En un sistema con tamaño de bloques de 4096 bytes existe un fichero de tamaño 8192. Suponiendo que en la cache de bloques no hay ninguna información sobre ese fichero, analice qué operaciones sobre la cache y sobre el disco produciría una petición de escritura sobre ese fichero que involucrase desde el byte 1.000 hasta el 10.000 (recuerde que la cache de bloques de UNIX no usa *write-through*).

Solución

a) En este apartado se utilizarán los operadores `/` y `%` sobre enteros con el mismo significado que tienen en C, o sea, división entera con truncamiento y resto de la división, respectivamente.
a.1) Dado un número de inodo **I** y teniendo en cuenta que el primer inodo es el 1, las fórmulas que calculan el bloque

de disco **B** donde se encuentra y el byte (**b**) dentro de dicho bloque son:

- `B= BI + ((I-1) x TI) / TB`
- `b= ((I-1) x TI) % TB`

a.2) A partir del número de bloque de datos **X**, las fórmulas que calculan el bloque de disco **B** donde se encuentra su información de estado y el bit **b** dentro de dicho bloque son:

- `B= MB + X / (TB x 8)`
- `b= X % (TB x 8)`

a.3) A partir del número de inodo **X**, las fórmulas que calculan el bloque de disco **B** donde se encuentra su información de estado y el bit **b** dentro de dicho bloque son:

- `B= MI + (X-1) / (TB x 8)`
- `b= (X-1) % (TB x 8)`

a.4) En primer lugar, se presenta la fórmula que calcula el bloque del fichero **BF** a partir de un determinado byte (**b**) del mismo:

- `BF= b / TB`

A continuación, se plantea la fórmula que, dado un número de bloque dentro del fichero **BF** y su inodo asociado **I**, determina el bloque de disco (**BD**) donde está almacenado. Esta fórmula se descompone en tres subfórmulas dependiendo de si el bloque dentro del fichero corresponde con un bloque directo, un indirecto simple o un indirecto doble, respectivamente.

- `Si BF < D -> I[BF]`
- `Si D £ BF < (D + TB/P) -> B(I[D])[BF-D]`
- `Si (D + TB/P) £ BF < (D + TB/P + (TB/P)`^{2}`) ->`
  - `B(B(I[D+1])[(BF-D-TB/P)/(TB/P)])[`*`(BF-D-TB/P)%(TB/P)`*`]`

Nótese en la última subfórmula, que corresponde con el caso donde el bloque **BF** está referenciado a través del puntero indirecto doble (`D+1`), que la expresión subrayada se usa para indexar dentro del bloque de primer nivel, mientras que la expresión en cursiva se usa para acceder al segundo.

Este proceso de traducción presenta una apreciable similitud con el proceso de traducción utilizado en un sistema con paginación. Para ello, hay que resaltar la siguiente equivalencia matemática:

- Si `X` es un número de `N` bits tal que `X=2`^{K} `->`
  - `Y%X = K` bits de menor peso de `X`
  - `Y/X = N-K` bits de mayor peso de `X`

Para ver la similitud, nótese, en primer lugar, que generalmente el valor de **TB** será una potencia de 2 (**TB** = 2^t) y, por tanto, suponiendo que la dirección de un byte del fichero **b** usa **n** bits, la primera fórmula de este apartado sería equivalente a:

- `BF= b/TB -> n-t` bits de mayor peso de `b`
  - Los `t` bits de menor peso corresponden con el desplazamiento dentro del bloque

Como se puede observar, es el mismo proceso que se realiza en un sistema de paginación.

En un sistema de paginación con un solo nivel, los bits de mayor peso (el equivalente a nuestro **BF**) se usan para indexar la tabla de páginas del proceso. En el caso del sistema de ficheros de UNIX, si **BF** se corresponde con un bloque directo o indirecto simple, este valor se usa para acceder a la tabla de punteros directos en el inodo o al bloque indirecto simple.

El acceso a bloques asociados al indirecto doble es similar a un sistema de paginación con dos niveles. En este caso, a partir del valor de **BF**, una vez ajustado (hay que restarle `D+TB/P` bloques que quedan cubiertos por los punteros directos y el indirecto simple), se calcula cómo acceder a los dos niveles de índices. Para ello, se usa el valor `TB/P`

t

que representa cuántas entradas caben en el bloque de índices de segundo nivel. Dado que **TB** es potencia de 2 (2^t) y, normalmente, también P (2^k), el valor de la división también lo será (**TB/P**=2^{t-k}=2^e). El cálculo es el siguiente:

- Para acceder al primer nivel se divide el valor de **BF** ajustado, que ocupa `n-t` bits, entre `TB/P -> (n-t)-e` bits de mayor peso de dicho valor ajustado.
- Para acceder al segundo nivel se calcula el resto entre el valor de **BF** ajustado y `TB/P -> e` bits de menor peso de dicho valor ajustado.

Nótese que cuando **BF** corresponde con el puntero indirecto doble es muy similar al proceso de traducción en un sistema con 2 niveles: La dirección del byte **b** dentro del fichero, después del ajuste debido a la existencia de punteros directos y el indirecto simple, se descompone en tres partes:

- Los `t` bits de menor peso corresponden con el desplazamiento dentro del bloque.
- Los `e` bits intermedios con el desplazamiento dentro del bloque de punteros de segundo nivel.
- Los `(n-t)-e` bits de mayor peso con el desplazamiento dentro del bloque de punteros de primer nivel.

Si se hubiera planteado un inodo más realista con un indirecto triple, se mantendría la analogía correspondiendo con un sistema de paginación de tres niveles.

Resumiendo, la traducción usando el inodo es similar a la usada en los sistemas de paginación, pero resultando un híbrido de sistemas con distintos niveles. En cualquier caso, es importante resaltar que una diferencia significativa es que el proceso de traducción de un sistema de memoria se realiza por hardware, ya que se debe aplicar a cada referencia a memoria, mientras que la traducción de ficheros la hace el sistema operativo, dado que sólo se aplica en los accesos a ficheros.

Hay que hacer notar, por último, que, gracias a que los tamaños de distintos elementos del sistema de ficheros son potencias de 2, el sistema operativo puede realizar más eficientemente las operaciones de división y resto requeridas por la traducción.

a.5) A partir de la petición `read(d,buf,tam)`, la lista de bloques de disco implicados se determina de la siguiente forma:

1) Se accede al BCP del proceso y dentro de éste a la tabla de descriptores abiertos, concretamente, a la posición *d* de esa tabla.
2) En esa posición se almacena qué entrada de la tabla intermedia de ficheros (tabla *filp*) corresponde con ese descriptor. En dicha entrada se almacena, entre otras cosas, la posición actual (*offset*) dentro del fichero y una referencia al inodo *I*, que estará almacenado en la tabla de inodos en memoria, puesto que el fichero se abrió previamente.
3) El rango de bytes que hay que leer correponden con el intervalo [*offset*, *offset+tam-1*]. Aplicando a los valores extremos de ese rango la primera fórmula del apartado anterior, se obtendrá el primer (BF_{ini}) y último bloque (BF_{fin}) del fichero que hay que leer, con lo que el intervalo resultante será: [BF_{ini}, BF_{fin}].
4) Por último, usando el inodo *I*, habrá que aplicar a cada uno de los bloques de esa lista la segunda fórmula del apartado anterior (la que relaciona bloques de fichero y bloques de disco), obteniendo la lista de bloques de disco.

b) Dado que se pretenden comparar dos formas de acceso a un fichero, nos vamos a centrar en la lectura propiamente dicha, puesto que la parte de apertura y cierre del fichero será común en ambas estrategias.
b.1) Hay que leer 21 bloques que producirán los siguientes accesos a la cache:

- Las 12 primeras lecturas, que implican los bloques directos, generan 12 accesos a la cache, con los consiguientes 12 fallos, puesto que la cache no tiene información del fichero.
- Las 9 lecturas restantes involucran al bloque indirecto simple. Por tanto, cada lectura implica dos accesos a la cache: uno para acceder al bloque indirecto y otro para leer el propio bloque de datos.
  - La primera de estas lecturas causa dos fallos
  - Las 8 restantes sólo generan fallo para acceder al bloque de datos ya que el indirecto ya está en la cache.

Nótese que la última lectura, aunque sea de sólo 20 bytes, implica los mismos accesos a la cache que las

lecturas anteriores.

En resumen:

- Número de accesos a la cache: 30
- Número de aciertos: 8; porcentaje: 8/30 = 26,67%
- Número de fallos (por tanto, accesos al disco): 22

b.2) Hay que leer 21 bloques que producirán los siguientes accesos a la cache:

- La primera lectura de 20 bytes provoca un acceso a la cache con un fallo que trae el primer bloque.
- Las 11 siguientes lecturas generan cada una dos accesos a la cache puesto que involucran dos bloques:
  - El primer bloque implicado se trajo en el acceso anterior produciéndose, por tanto, un acierto.
  - El segundo bloque requerido provoca un fallo ya que no está en la cache.
- La decimotercera lectura implica a dos bloques que corresponden a rangos de cobertura del inodo diferentes:
  - El primer bloque implicado corresponde con un bloque directo, que se trajo en el acceso anterior produciéndose, por tanto, un acierto.
  - El segundo bloque requerido corresponde con un bloque indirecto, produciéndose, por tanto, dos accesos a la cache (el bloque indirecto y el propio dato), generando ambos un fallo.
- Cada una de las 8 lecturas restantes involucra a dos bloques generando 4 accesos:
  - El primer bloque implicado provoca un primer acierto en el acceso al bloque indirecto y un segundo acierto al acceder al propio bloque de datos, ya que se trajo en el acceso anterior.
  - El segundo bloque requerido provoca un primer acierto en el acceso al bloque indirecto, pero un fallo al acceder al propio bloque de datos.

En resumen:

- Número de accesos a la cache: 58
- Número de aciertos: 36; porcentaje: 36/58 = 62,07%
- Número de fallos (por tanto, accesos al disco): 22

Comparando los resultados, se puede ver que el número de acceso al disco es el mismo, lo cual es lógico puesto que hay que traer la misma información, y el número de llamadas al sistema es también igual. Sin embargo, se produce un número considerablemente mayor de accesos a la cache con la segunda estrategia, lo que conlleva un peor tiempo de acceso al fichero. Esto se notaría sobretodo en sistemas con discos de altas prestaciones donde la sobrecarga del acceso a la cache sería más significativa.

c) Aplicando la primera fórmula del apartado **a.4**, el rango del fichero afectado por la lectura corresponde con los tres primeros bloques del mismo, generando, por tanto, tres accesos a la cache:

- Primer bloque (bytes desde 1000 hasta 4095). Como se trata de una escritura incompleta, que no afecta a todo el bloque, hay que leer el bloque antes de modificarlo. Puesto que no está en la cache, hay que leerlo del disco.
- Segundo bloque (bytes desde 4096 hasta 8191). Al tratarse de una escritura completa, que involucra a todo el bloque, no es necesario leer el bloque antes de sobre-escribirlo. Simplemente, se reserva un buffer en la cache y se copian en él los datos de la escritura.
- Tercer bloque (bytes desde 8192 hasta 10000). Esta escritura causa que crezca el fichero. Hay que buscar en el mapa de bloques uno libre, reservarlo y asignarlo al fichero en su inodo. Nótese que, aunque se trata de una lectura incompleta, no hay que leer el nuevo bloque del disco puesto que su contenido previo no es significativo. Como en el caso anterior, basta con reservar un buffer en la cache y copiar en él los datos de la escritura.

Hay que resaltar que, dado que no se usa *write-through*, las escrituras en los tres bloques no causan escrituras inmediatas en el disco, sino que éstas se diferirán hasta que el bloque sea expulsado o se cumpla el plazo de volcado periódico. Asimismo, nótese que en cualquiera de los tres accesos, a la hora de reservar un buffer libre en la cache, podría haberse requerido una expulsión de otro bloque que estuviese modificado, produciéndose, en ese caso, una escritura en disco.