Sistemas Operativos 4º Semestre. Grados II y MI

Primer Parcial. Sistema de Ficheros. 12 de Marzo de 2015.

Dispone de 60 minutos. Publicación: el 26 de Marzo. Revisión: el 10 de Abril.

Ejercicio único

Los siguientes apartados están relacionados.

A) (2 puntos) Implemente en C y para UNIX el mandato `incrementa` *`archivo`*, que:

A1) Abre el archivo indicado como argumento y redirige la entrada y la salida estándar a dicho archivo.

A2) Seguidamente, a través de los descriptores estándar, incrementa en 5 el entero binario (`int`) contenido en la posición 3 GiB del archivo.

SOLUCIÓN:

A1)

```
 1      int main(int argc,char*argv[])
 2      {
 3          int fch = open(argv[1],O_RDWR|O_CREAT,0666);
 4          if (fch==-1) { perror("open"); exit(1); }
 5          dup2(fch,0);
 6          dup2(fch,1);
 7          close(fch);
```

A2)

```
 8      #define POS (3UL*(1<<30))
 9          int ofs = lseek(0,POS,SEEK_SET);
10          if (ofs==-1) perror("lseek1");
11          int val = 0;
12          int cnt = read(0,&val,sizeof(int));
13          if (cnt==-1) perror("read");
14              ofs = lseek(1,-cnt,SEEK_CUR);
15          if (ofs==-1) perror("lseek2");
16              val += 5;
17              cnt = write(1,&val,sizeof(int));
18          if (cnt==-1) perror("write");
19          return 0;
20      }
```

B) (2 puntos) Según su implementación, ¿cómo se comportaría `incrementa` en los siguientes casos?:

B1) Si el archivo indicado fuese un fichero especial orientado a carácter, (ej. un terminal).

B2) Si el archivo indicado fuese un fichero normal con tamaño inicial 2 MiB. ¿Qué tamaño al final?

SOLUCIÓN:

B1) Suponiendo que el terminal pudiese ser abierto, las llamadas `lseek` fallarían (no existe el concepto de "posición" sobre un terminal) y las llamadas `read` y `write` estarían dialogando a través del terminal con un usuario, que no puede introducir información en binario, sino sólo texto.

B2) La primera llamada `lseek` se posicionaría más allá del tamaño del fichero, con lo que la llamada `read` devolvería `0` bytes leídos. Tal y como se ha implementado, `val` valdría `0`, la segunda llamada a `lseek` no nos movería de la posición `POS` y la llamada `write` escribiría el entero binario de valor `5` en la posición indicada. El tamaño final del fichero serían `POS` más `sizeof(int)` bytes, y el espacio intermedio desde los 2 MiB hasta los 3 GiB se habría rellenado con bytes nulos.

C) (2 puntos) Considerando que la cache de bloques del servidor de ficheros está inicialmente vacía y utilizando los mismos datos y bajo las mismas suposiciones del apartado **D**, conteste a las siguientes preguntas, donde nos referimos a las llamadas necesarias para implementar el mandato del apartado **A** (que incrementa el entero binario contenido en la posición 3 GiB de un archivo).

C1) Razone, ¿cuántos accesos a disco son necesarios para realizar la llamada `read` (del apartado **A**)?

C2) Razone, ¿cuántos accesos a disco son necesarios para realizar la llamada `write` (del apartado **A**)?

SOLUCIÓN:

C1) Para acceder a la posición 3 GiB de este archivo habrá que acceder a la agrupación lógica número (3*2^30 (B) / 2^13 (B/agrp) = 3*2^17 (agrp)) 393216 de su inodo.
Según las suposiciones realizadas en el apartado D (direcciones de 32 bits y 2^11 (dirs/argp)), habrá que acceder a través del puntero doble indirecto del inodo. Esto supone acceder a tres agrupaciones. Como la cache de bloques está inicialmente vacía, serán accesos al disco.

C2) Para la correspondiente operación de escritura sobre la misma posición del archivo, dado que la agrupación afectada (así como las agrupaciones de indirección necesarias para localizar esta) ya estarán en la cache de bloques, no será necesario acceder al disco, bastará con realizar esta modificación sobre la cache de bloques, esto es, en memoria. La posterior escritura desde la cache hacia el disco de esta información contenida en la cache y ahora sucia (agrupación con número lógico 393216) sucederá más tarde, dependiendo de la política de gestión de esta cache.

D) (2 puntos) Considere que el sistema de ficheros subyacente es de tipo UNIX (basado en inodos), con agrupaciones de 8 KiB, para un disco de medio TiB y archivos de tamaño medio igual a una agrupación. Justifique las suposiciones que considere necesarias.

D1) Exprese la capacidad de direccionamiento de un inodo en este sistema. Dé el resultado en bytes.

D2) Razone, ¿cuánto ocupará el mapa de bits de inodos en este sistema? Indique cálculos y unidades.

SOLUCIÓN:

D1) Lo primero hace falta saber es el tamaño de las direcciones a agrupación.
Dado que: 2^39 (B/disco) / 2^13 (B/agrp) = 2^26 (argp/disco), bastan 26 bits para numerar todas las agrupaciones, luego usaremos direcciones de 32 bits.
En una agrupación de indirección cabrán: 2^13 (B/agrp) / 2^2 (B/dir) = 2^11 (dirs/argp)
Suponiendo un inodo con 10 punteros directos y 3 niveles de indirección, se podrán direccionar:
[10 (p.dir) + 2^11 (s.ind) + 2^22 (d.ind) + 2^33 (t.ind)] (agrupaciones).
Para pasar a bytes multiplicamos por el tamaño de la agrupación en bytes.
En números redondos serían 2^(33+13) (B) = 64 TiB y pico.

D2) Preparamos un inodo por cada posible archivo de tamaño medio, esto es, 2^26 (inodos).
El mapa de bits necesita un bit por inodo, luego ocupará: 2^26 (bits) / 2^3 (bits/B) / 2^13 (B/agrp) = 2^10 (agrupaciones).

E) (2 puntos) Sea la siguiente visión parcial del contenido del sistema de ficheros:

NUM	T	U	G	O	USER	GROUP	PATH (RUTA)
1	d	rwx	r-x	r-x	root	root	/
2	d	rwx	rwx	rwt	root	root	/tmp/
3	-	rwx	rw-	r--	yoda	jedi	/tmp/datos.bin
4	d	rwx	r-x	r-x	root	root	/home/
5	d	rwx	r-x	---	r2d2	robot	/home/r2d2/
6	l	rwx	rwx	rwx	r2d2	robot	/home/r2d2/temporal -> /tmp/
7	-	rwx	r-s	---	r2d2	jedi	/home/r2d2/incrementa

Donde: NUM es el número de ruta; T es el tipo de objeto; U, G y O son los grupos de permisos (*user*, *group* y *other*); y la notación `A -> B` indica que el enlace simbólico `A` apunta a la ruta `B`.

Considere que el usuario `r2d2` (del grupo `robot`) invoca, desde su *home*, el siguiente mandato:

```
./incrementa   temporal/datos.bin
```

Para cada uno de los siguientes casos, conteste detallando cómo el sistema operativo decodificará la ruta indicada. Muestre: (número de) ruta, grupo de permisos y permiso concreto, que se validará a cada paso.

E1) ¿Podrá ponerse en ejecución el mandato indicado? ¿Qué identidad tendrá el proceso resultante?

E2) ¿Podrá el proceso abrir el archivo indicado? Detalle cada paso de la decodificación de la ruta.

SOLUCIÓN:

E1) Sí podrá ejecutarse la ruta relativa: `./incrementa`
Detalle: [`./ 5Ux`] [`incrementa 7Ux`].
Tendrá identidad efectiva de grupo `jedi`, por [`incrementa 7Gs`].

E2) Sí podrá abrirse la ruta relativa: `temporal/datos.bin`
Detalle: [`./ 5Ux`] [`temporal/ 6Tl`] (seguimos con `/tmp/`) [`/ 1Ox`] [`tmp/  2Ox` (la `x` bajo la `t`)] (seguimos con `datos.bin`) [`datos.bin 3Grw`].