SOA2 (30/05/2023)

Nombre:
DNI:

Segundo control de teoría

Contesta todas las preguntas justificando brevemente tu respuesta. Una respuesta sin justificar se considerará como incorrecta.

1. *(2 puntos) Estructuras del sistema de ficheros*

Imagina que eliminamos del sistema de ficheros la Tabla de I-nodes, y guardamos en cada entrada de la TFA el número de Inodo y el puntero a descriptor de dispositivos correspondientes. Supón que no existe ningún gestor. Contesta a las siguientes preguntas:

a. Con esta modificación, ¿se mantiene el acceso compartido a los dispositivos lógicos?

Si, ya que simplemente se crearán nuevas entradas en la TFA y se podrá acceder desde operaciones distintas al mismo dispositivo lógico?

b. Con esta modificación, ¿se mantiene el acceso concurrente a los dispositivos lógicos?

Si, debido a lo que he mecionado en la respuesta a, y gracias al puntero de lectura / escritura, se podra acceder simultaneamente al mismo dispositivo

c. ¿Qué modificaciones se tienen que hacer en la llamada al sistema open?

Se salta el paso de crear una nueva entrada en la tabla de inodos, siempre que se haga un open se creará una nueva entrada en la TFA, pero si es a un dispositivo que previamente se había hecho open, los punteros a *inodo, y *descriptor de dispositivo, apuntaran al mismo sitio siempre (para ese dispositivo)

d. ¿Qué modificaciones se tienen que hacer en la llamada al sistema read?

Accede a la TFA y de la TFA accede directamente al puntero de descriptor de dispositivo, en vez de pasar por la tabla de inodos.

e. El sistema de ficheros evita que se borre un dispositivo lógico si éste está en uso. ¿Cómo se haría esta comprobación?

La TFA tiene un campo que es el numero de referencias totales que tiene en Tablas de Canales. Cada vez que se hace open se incrementa, y cuando se hace close decrementa (o cuando se clonan / matan procesos). Si aun tiene referencias no se puede borrar este dispositivo lógico porque sigue en uso.

SOA2 (30/05/2023)

Nombre:
DNI:

2. *(3 puntos) Sistema de ficheros:*

Tenemos un sistema de ficheros cuyo tamaño de bloque de datos es de 1024 bytes, equivalente a 4 sectores de disco. El sistema de ficheros contiene 8192 Inodos y 4096 bloques de datos. Siempre se utilizarán Inodos para guardar la información relativa a un fichero y el número de Inodos será estático (siempre habrá 8192). Los índices a bloque ocupan 4 bytes.

En el caso de que todos los ficheros tengan exactamente 1024 bytes, indica, justificándolo, el número de ficheros máximo que podemos crear si el mecanismo de asignación de bloques a ficheros es:

a) Continuo

(4096 bloques * 1024 B/bloque)/ 1024 B/fichero = 4096 ficheros

b) Encadenado (sin tabla)

(4096 bloques * 1024 B/bloque)/ 1024 + 8 B/fichero = ~ 4064 ficheros
Esos 8 bytes son los del puntero al siguiente bloque

c) Encadenado con tabla Asumo que la FAT esta en disco y ocupa bloques

La tabla ocupa 4B * 4096 bloques = 16 384 B, ocupa 16 bloques de datos
((4096-16) * 1024)/1024 = 4080 archivos.

d) Indexado Asumo que se usan los BD como BIs y que los BIs no son aparte

En 1 bloque de indices caben = 1024/4 indices = 256 indices. pero uno se quita pq apunta al siguiente bloque de indices, 255.
4096 bloques de datos necesitan 17 bloques de indices para indexarse. Entonces:
((4096-17) * 1024)/1024 = 4079 archivos.

e) Indexado multinivel Asumo que se usan los BD como BIs y que los BIs no son aparte

Si todos los ficheros ocupan 1024 bytes, se consideran ficheros pequeños, ya que ocupan un bloque cada uno. En el inodo hay un puntero al boque que ocupan.
(4096 bloques * 1024 B/bloque)/ 1024 B/fichero = 4096 ficheros

f) ¿Cuántos índices caben en un bloque de índices?

Caben 1024 B Bloque /4B indice = 256 Indices/Bloque

Nombre:
DNI:

g) ¿Hasta que tamaño de fichero la asignación encadenada sin tabla ocupará menos espacio en disco que la indexada?

Hasta 1024-8 = 1016 B por fichero

h) ¿Cuántos accesos a disco se tendrán que realizar para acceder al offset 1063004 en este sistema de ficheros si la asignación es indexada multinivel con 10 índices directos a bloques de datos, 1 indirecto y 1 doblemente indirecto dentro del Inodo?

Calculamos niveles:
1) 10*1024 = 10240bytes, del 0 a 10 239 Nivel 1
2) 256 * 1024 = 262 144 bytes, del 10 240 al 272 383
3) 65 536 * 1024 = 67 108 864 bytes, del 272 384 al 67 381 248

Tendrá que cojer el 3 puntero
Traerse el bloque de indices
Traerse el tercer bloque de dentro
En ese bloque de indices, la cuarta entrada apunta al bloque de datos que contiene el offset = 3 accesos

i) Indica, cual será el tamaño de los mapas de bits de Inodos y bloques de datos libres en esta partición.

Asumiendo que cada bitmap ocupa solo un bit por inodo/bloque de datos:
Bitmap inodos 8192/ 8 = 1024 Bytes
Bitmat bloque de datos 4096/ 8 = 512 Bytes

j) Indica cual será el tamaño de la FAT si esta partición utiliza asignación encadenada en tabla

La tabla ocupa 4B * 4096 bloques = 16 384 B

3. (3 puntos) Syncro

Queremos implementar el código para un dispositivo que se llamará Syncro. Este dispositivo permitirá la sincronización entre threads de tal forma que cuando un thread lea este dispositivo, se quedará bloqueado hasta que alguien escriba en este dispositivo. En el caso de que haya más de un thread bloqueado, desbloqueará a uno de ellos. Tanto read como write devolverán 0.
La sucesión de llamadas a funciones que se hace hasta llegar al código del driver Syncro, para la llamada al sistema read, es:

1.- int sys_read(int fd, void *buffer, size_t size)
2.- int read_vfs(struct Inode *Inode, int offset, void *buffer, size_t size)
3.- int read_syncro(struct opaque* data, int offset, void *buffer, size_t size)

Por ahora supondremos que no tenemos ningun gestor.

SOA2 (30/05/2023)

Nombre:
DNI:

a) Enumera las estructuras del sistema de ficheros a las que accede sys_read

Tabla de Canales del proceso -> Tabla de ficheros abiertos -> Tabla de inodos
Le da el puntero al inodo a read_vfs, read_vfs

b) ¿Qué campos debería contener la estructura struct opaque? ¿Cómo y dónde se inicializan?

```
struct opaque {
    *descriptor_disp;
    ?? Nose tio
};
```

c) Escribe el código de read_syncro

d) Escribe el código de write_syncro

e) ¿Cómo obtiene read_vfs la dirección de la función del driver read_syncro?

Ahora añadimos un único gestor entre el virtual file system y el driver syncro. El virtual file system no tiene ningun gestor asociado.

f) ¿Se tiene que modificar el código de la función read_vfs?

g) ¿Qué estructuras de datos se han tenido que añadir para poder enviar peticiones a este gestor?

h) A parte del semáforo del gestor, ¿Cuántos semáforos se han tenido que añadir para con este nuevo gestor?

i) ¿Para qué vale el semáforo del gestor? ¿Cómo se inicializa?

j) Escribe el código de este gestor.

k) Dado el funcionamiento de este dispositivo, ¿sería necesario tener un gestor?

4. (2 puntos) Rendez-vous

Un Rendez-vous es un mecanismo que sirve para intentar aumentar la posibilidad de que dos partes del código se ejecuten en paralelo. Normalmente se hace de la siguiente forma para 2 threads:

```
sem_t sem1, sem2;
sem_init(&sem1, 0);
sem_init(&sem2, 0);
```

T1	T2
sem_post(&sem1);	sem_post(&sem2);
sem_wait(&sem2);	sem_wait(&sem1);
código paralelo	código paralelo

SOA2 (30/05/2023)

Nombre:
DNI:

En un mundo ideal, con un ordenador con varios procesadores, T1 y T2 con el Rendez vous, ejecutarían “código paralelo” a la vez. Pero como habéis visto durante este cuatrimestre, los sistemas operativos no tienen nada de “ideal”.

a) (0.5 puntos) ¿Por qué no se puede asegurar con un Rendez-vous que la línea “código paralelo” en los 2 threads se ejecute a la vez?

b) (0.5 puntos) ¿Se podrían hacer primero los sem_wait y después los sem_post en cada uno de los threads?

c) (1 punto) Escribe el código para hacer un Rendez-vous entre 3 threads.