Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas

Sistemas Operativos
Segundo parcial (con solución)
Segundo cuatrimestre de 2013

Aclaraciones

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Realice cada ejercicio en hojas separadas y NO utilice ambas carillas de la hoja en las respuestas a los ejercicios de semáforos.
- Cada ejercicio se califica con Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente. El parcial se debe aprobar con 2 ejercicios bien y a lo sumo 1 mal.
- El parcial NO es a libro abierto; sin embargo, se permite tener dos hojas A4 con apuntes.
- Por favor entregar esta hoja junto al examen.
- **Importante:** Justifique sus respuestas.

Ejercicio 1) Se desea implementar el driver de una controladora de una vieja unidad de discos ópticos que requiere controlar manualmente el motor de la misma. Esta controladora posee 3 registros de lectura y 3 de escritura. Los registros de escritura son:

- `DOR_IO`: enciende (escribiendo 1) o apaga (escribiendo 0) el motor de la unidad.
- `ARM`: número de pista a seleccionar.
- `SEEK_SECTOR`: número de sector a seleccionar dentro de la pista.

Los registros de lectura son:

- `DOR_STATUS`: contiene el valor 0 si el motor está apagado (o en proceso de apagarse), 1 si está encendido. Un valor 1 en este registro no garantiza que la velocidad rotacional del motor sea la suficiente como para realizar exitosamente una operación en el disco.
- `ARM_STATUS`: contiene el valor 0 si el brazo se está moviendo, 1 si se ubica en la pista indicada en el registro `ARM`.
- `DATA_READY`: contiene el valor 1 cuando el dato ya fue enviado.

Además, se cuenta con las siguientes funciones auxiliares (ya implementadas):

- `int cantidadSectoresPorPista()`: Devuelve la cantidad de sectores por cada pista del disco. El sector 0 es el primer sector de la pista.
- `void escribirDatos(void * src)`: Escribe los datos apuntados por `src` en el último sector seleccionado.
- `void sleep(int ms)`: Espera durante `ms` milisegundos.

Antes de escribir un sector, el driver debe asegurarse que el motor se encuentre encendido. Si no lo está, debe encenderlo, y para garantizar que la velocidad rotacional sea suficiente, debe esperar al menos 50 ms antes de realizar cualquier operación. A su vez, para conservar energía, una vez que finalice una operación en el disco, el motor debe ser apagado. El proceso de apagado demora como máximo 200 ms, tiempo antes del cual no es posible comenzar nuevas operaciones.

a) Implemente la función `write(int sector, void* data)` del driver, que escriba los datos apuntados por `data` en el sector en formato LBA indicado por `sector`. Para esta primera implementación no use interrupciones.
b) Modifique la función del inciso anterior utilizando interrupciones. La controladora del disco realiza una interrupción en el `IRQ 6` cada vez que los registros `ARM_STATUS` o `DATA_READY` toman el valor 1. Además, el sistema ofrece un *timer* que realiza una interrupción en el `IRQ 7` una vez cada 50 ms. Para este inciso no puede utilizar la función `sleep`.

Solución. [1]

A)

```
int write(int sector, void* data){
int sec_por_pista = cantidadSectoresPorPista(); // cuantos sectores tengo por cada pista
OUT(ARM, sector / sec_por_pista); // haciendo la división entera obtento el número de pista
OUT(SEEK_SECTOR, sector % sec_por_pista); // con el módulo obtengo el sector que quiero escribir

// tengo que esperar a que el brazo esté en posición
while(IN(ARM_STATUS) != 1){
sleep(1); // no hago nada, espero a que se posicione
}

// tengo el brazo en posición, enciendo el motor
if(IN(DOR_STATUS) == 0)
OUT(DOR_IO, 1); // enciendo el motor
sleep(50); // duermo 50 ms para asegurarme que el motor tiene la velocidad rotacional necesaria
   // si estaba encendido el motor desde antes lo hago también por las dudas

// ya puedo escribir
escribirDatos(data);

// tengo que esperar que se haya escrito todo bien
while(IN(DATA_READY) != 1){
sleep(1); // espero que se haya pasado la data
}

// terminé de escribir, tengo que apagar el motor
OUT(DOR_IO, 0); // le indico al motor que se apague
sleep(200); // duermo 200 ms hasta asegurarme que se haya apagado el motor

return IO_OK; // terminé bien
}
```

B)

```
mutex arm = 0;
mutex transfer = 0;
mutex time = 0;

int driver_init(){
if(request_irq(6, arm_and_data) == IRQ_ERROR) // mapeo la IRQ 6 con el handler
return IO_ERROR;
if(request_irq(7, temp) == IRQ_ERROR) // mapeo la IRQ 6 con el handler
return IO_ERROR;
return IO_OK;
}

int driver_remove(){
free_irq(6); // libero la IRQ 6 del handler
free_irq(7); // libero la IRQ 7 del handler
return IO_OK;
}

int write(int sector, void* data){
int sec_por_pista = cantidadSectoresPorPista(); // cuantos sectores tengo por cada pista
```

```
OUT(SEEK_SECTOR, sector % sec_por_pista); // con el módulo obtengo el sector que quiero escribir
OUT(ARM, sector / sec_por_pista); // haciendo la división entera obtento el número de pista

// tengo que esperar a que el brazo esté en posición
// AHORA ME LO AVISAN POR INTERRUPCIÓN, LA MANEJA EL HANDLER, ESPERO EL MUTEX
arm.wait(); // espero que el brazo esté

// tengo el brazo en posición, enciendo el motor
if(IN(DOR_STATUS) == 0)
OUT(DOR_IO, 1); // enciendo el motor
// duermo 50 ms para asegurarme que el motor tiene la velocidad rotacional necesaria
// si estaba encendido el motor desde antes lo hago también por las dudas
// AHORA ME LO AVISAN POR INTERRUPCIÓN, LA MANEJA EL HANDLER, ESPERO EL MUTEX
int i = 0;
while(i < 2){
timer.wait(); // espero que interrumpa una vez el timer, pasaron 50 ms
i++;
}

// ya puedo escribir
escribirDatos(data);

// tengo que esperar que se haya escrito todo bien
// AHORA ME LO AVISAN POR INTERRUPCIÓN, LA MANEJA EL HANDLER, ESPERO EL MUTEX
transfer.wait(); // espero se haya pasado la data

// terminé de escribir, tengo que apagar el motor
OUT(DOR_IO, 0); // le indico al motor que se apague
// duermo 200 ms hasta asegurarme que se haya apagado el motor
// AHORA ME LO AVISAN POR INTERRUPCIÓN, LA MANEJA EL HANDLER, ESPERO EL MUTEX
i = 0;
while(i < 5){
timer.wait(); // espero que interrumpa una vez el timer, pasaron 50 ms
i++;
} // acá me interrumpieron 4 veces, 4*50 = 200 ms como quería

return IO_OK; // terminé bien
}

void arm_and_data(){
if(IN(ARM_STATUS) == 1)
arm.signal();
if(IN(DATA_READY) == 1)
transfer.signal();
}

void temp(){
timer.signal();
}
```

Ejercicio 2)

a) Suponga un sistema operativo con un administrador de memoria paginada de 1 nivel cuyo tamaño de página es de 8KB. Se conoce que el sistema operativo funciona sobre un procesador cuyo tamaño de palabra es de 64 bits y el direccionamiento es de a palabra. Se pide:

- Calcule cuántos bits se destinan para el offset dentro de la página y cuántos para direccionar a las páginas.
- Calcule el tamaño máximo que puede llegar a alcanzar la tabla de página de un proceso, suponiendo que cada entrada de la tabla de páginas ocupa 128 bits.
- Calcule cuántas páginas puede llegar a ocupar como máximo la tabla de páginas.
- Dado un proceso en ejecución que requiere 7MB, calcule cuántas entradas de la tabla de páginas están siendo empleadas por dicho proceso en el sistema.

b) Considere un sistema de paginación bajo demanda en el que un proceso que tiene asignados 4 marcos de página genera la siguiente secuencia de referencias a páginas: 4,2,4,1,6,3,2,5,6,4,1,3,5,3. Indicar qué accesos producirían un fallo de página cuando se utiliza cada una de las políticas de reemplazo local FIFO, LRU y óptima.

Solución. [2]

A) El tamaño de página es de 8KB = 2^{13}B. Si direcciono a byte, tengo 13 bits para la zona de offset. Si direcciono a palabra de 64 bits (8B), tengo $2^{13}/2^3 = 2^{10}$ bits de offset.

Conocido el offset (10 bits) y el tamaño de la palabra (64 bits) podemos obtener los bits que ocupan la zona de número de página. 64 bits - 10bits = 54 bits.

B) Nuestra tabla de páginas tendrá como máximo tantas entradas como páginas pueda tener, por lo que podría tener 2^{54} entradas. Si sabemos que cada entrada ocupa 128 bits (16 B = 2^4B) el tamaño máximo de la tabla de páginas será $2^{54} * 2^4 = 2^{58}$ B = 256 Petabytes.

C) Sabiendo el tamaño máximo de la tabla de páginas y el tamaño de página, podemos saber cuántas páginas necesita. 2^{58} B / 2^{13} B = 2^{45} páginas = 32 Tera Páginas.

D) Si el proceso ocupa 7MB ($7 * 2^{20}$B) y cada página de la tabla de páginas ocupa 8 KB (2^{13}B) tendremos, $7 * 2^{20}$B / 2^{13}B = $7 * 2^7$ entradas de la tabla de páginas están siendo ocupadas.

Ejercicio 3) En el centro de datos de una empresa de ventas por internet comenzaron a preocuparse por el volumen de datos que manejan y su importancia. Actualmente utilizan el sistema de archivos distribuido BardoFS en el cual cada usuario guarda sus archivos en el disco de su PC y los comparte por Dropbox a todos sus colegas. Preocupados por el caos generado por los discos en mal estado y los nuevos virus, decidieron convocarnos para buscar una solución a sus problemas.

Estos son los requisitos que nos pidieron:

- Si fallan uno o dos discos rígidos, todo debe seguir funcionando. No debe tener que apagarse el sistema.
- El área de Recursos Humanos quiere 10 TB libres para mantener sus archivos en un sistema de archivos encriptado EFS, mientras que el área Diseño quiere 50 TB y usar NTFS. Finalmente el área Sistemas quiere utilizar 12 TB en EXT4.
- Físicamente debe ser un único servidor de archivos ubicado en una habitación a prueba de incendio. El sistema operativo del servidor requiere 1 TB de espacio.
- No necesitamos discos separados para cada partición, podemos usar particiones virtuales del disco duro. No les interesa a las distntas áreas tener sus particiones físicamente separadas.
- Todos los viernes se debe hacer una copia de seguridad, pero a su vez, se quiere poder restaurar los archivos modificados entre el día actual y el último viernes, aunque sólo para la semana en curso.

Estos son los recursos que pueden adquirirse:

- Servidor (Motherboard + CPU + Memoria + Gabinete).
- Controladora RAID (Configurable).
- HDD 1 TB 7200 RPM Sata III.
- HDD 2 TB 7200 RPM Sata III.
- Unidad de Cinta.
- Cinta de 10 TB.

Diseñe una solución que satisfaga los requisitos y a su vez minimice la cantidad de recursos e insumos requeridos. Justifique como satisface cada requisito.

Solución. [3]

Asumo que:

- No se puede crear un RAID 6 con discos de distinta topología. Esto es por el necesario mantenimiento de los bits de paridad para cada bloque de disco (sin lo cual es chueco).
- La implementación de RAID 6 no tiene un límite máximo de discos.
- Queremos que el sistema no se caiga JAMÁS por la caída de dos discos, no que sea altamente probable.
- No necesitamos discos separados para cada partición, podemos usar particiones virtuales del disco duro. Más aún, todo el RAID lo vemos como un solo disco. No les interesa a los departamentos tener sus particiones físicamente separadas.
- Las copias de seguridad de todos los lunes son totales.
- Las copias de seguridad de las modificaciones diferenciales entre el día y el viernes anterior entran en una sola cinta de 10 TB.
- Dado que no se da costos de cada hardware, lo que nos interesa minimizar es la cantidad de insumos a utilizar.

Utilizo un Servidor, una controladora de RAID 6 (para que podamos tener caída de hasta dos discos) y una unidad de cinta. Veamos cuantos discos y cintas necesito.

En total necesito 73 TB (contando Sistema Operativo) de espacio efectivo. Asumiendo que todos los discos deben tener la misma topología, utilizo todos discos de 2TB. Por lo tanto, redondeo para arriba y utilizo $\lceil \frac{73}{2} \rceil = 37$ discos duros de 2 TB. Necesito además dos discos de 2TB para paridad, para un total de 39 discos.

Además, utilizo $\lceil \frac{73}{10} \rceil = 8$ cintas de backup para cada copia de seguridad de los lunes, para hacer una copia total, y luego utilizo una cinta adicional para hacer una copia diferencial del día de hoy con el

viernes anterior. No es necesario tener una cinta por cada día de backup diferencial, puedo reescribir la anterior, pero como no quiero reescribir en la misma cinta donde tengo el último backup diferencial por si llega a haber un problema con la cinta y me quedo sin backup diferencial, utilizo en total dos cintas.

Necesito por lo tanto 39 discos, 2 cintas de backup diferencial, más 8 cintas por cada semana que quiera guardar los backups.

Ejercicio 4) Se tiene un disco rígido de 2 platos, dos caras por plato, 2048 pistas y 4096 sectores de 1 KB por pista. Se desea dar formato al disco usando un sistema de archivos de uso específico llamado HashFS basado en FAT. La idea es que no existen directorios ni archivos. Dado un path, se calcula el *Hash* del nombre y este indica cual es el archivo buscado. En resumen, este sistema de archivo cuentas con dos tablas:

- Una única FAT que guarda las entradas correspondientes al próximo bloque, indicando el final de un archivo cuando estos valores coinciden.
- Una única tabla de Hash que contiene los números de bloques iniciales y el tamaño en bytes para cada hash posible.

La novedad, es que este sistema de archivos permite configurar los siguientes elementos:

- Tamaño del bloque: 2, 4 o 8 sectores.
- Tamaño de identificadores de bloque: 8, 16, 24 o 32 bits.
- Tamaño del hash: 8, 16, 24 o 32 bits.

a) Suponiendo que se configura con 2 sectores por bloque, identificadores de bloque de 16 bits, y hash de 16 bits. ¿Cuál es el tamaño que ocupa la FAT? ¿Cuál es el tamaño de la tabla de archivos? ¿Cuál es el espacio que queda en disco para archivos?
b) Sabiendo que se desea maximizar la cantidad de archivos que el sistema soporta y que, además, en promedio los archivos tendrán un tamaño de 1 KB. ¿cuál sería la configuración óptima del sistema de archivos? Justifique.
c) ¿Cómo lo configuraría si el promedio de tamaño de archivos es de 16 Mb? Justifique.

Solución. [4]

A) Mi disco tiene 2 platos * 2 caras * 2048 (2^{11}) pistas * 4096 (2^{12}) sectores. Es decir, $2*2*2^{11}*2^{12} = 2^{25}$ sectores en total. Y un tamaño total de 32GB (2^25*1KB).

La tabla FAT tiene, por cada entrada: dos identificadores de bloques. Si se configura con 2 sectores por bloque, tengo 2^{24} bloques. Sin embargo, como mi identificador de bloque es de 16 bits, sólo puedo direccionar a 2^{16} bloques. Por lo tanto, la cantidad de entradas de mi tabla FAT es de 2^{16} y cada entrada es de 32 bits (2*16bits). Entonces el tamaño total de mi tabla FAT es de 256KB.

La tabla HASH tiene, por cada entrada: un hash, un identificador de bloque y un tamaño en bytes (que asumimos un int de 4B). El tamaño máximo de entradas está dado por el tamaño del hash (2^16). Entonces el tamaño total de mi tabla HASH es de 2^16* (16 bits + 16bits + 8B) = 768 KB.

B) Como me gustaría maximizar la cantidad de archivos, y la cantidad de archivos que puedo referenciar está limitada por el tamaño del hash. En principio, trato de elegir el hash más grande. Si eligo 32 bits de hash, puedo referenciar a 2^{32} archivos. Pero como mi disco tiene 32GB, y cada archivo en promedio va a ocupar 2KB (el enunciado dice 1KB, pero el tamaño mínimo de sectores por bloque es de 2 sectores de 1KB cada uno), entonces me basta con poder referenciar a 32GB / 2KB = 2^{24} archivos. Por lo tanto, utilizo un hash de 24 bits.

Para el tamaño de bloque, tengo 2^{25} sectores y cada bloque tiene dos sectores, entonces necesito poder referenciar a $2^{25}/2$ bloques. Entonces elijo un identificador de 24bits.

En resumen, tengo 2 sectores por bloque, 24 bits para el hash y 24 bits para el identificador de bloque.

C) Mi disco va a necesitar direccionar aproximadamente a 32GB / 16Mb = 2^{14} archivos. Por lo tanto, utilizo un identificador de hash de 16 bits.

Como los archivos son en promedio de 16Mb, elijo 8 sectores por bloque que es el máximo permitido. Por lo tanto, voy a necesitar referenciar a $2^{25}/2^3 = 2^{22}$ bloques. Lo cual me basta con un identificador de 24 bits para el bloque.

En resumen, tengo 8 sectores por bloque, 16 bits para el hash y 24 bits para el identificador de bloque.