PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

4ta práctica (tipo a)
(Segundo semestre de 2016)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. F. Solari A.

Duración: 1 h. 50 min.
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (7 puntos – 35 min.) *(Memoria virtual – algoritmos)* Dada la cadena de referencia de páginas siguiente:

0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

a) (2 puntos – 10 min.) Dada una asignación de 3 marcos de página, y suponiendo que la memoria está inicialmente descargada, ¿cuántos fallos de página se producirán con el algoritmo óptimo?

b) (2 puntos – 10 min.) Dada una asignación de 3 marcos de página, y suponiendo que la memoria está inicialmente descargada, ¿cuántos fallos de página se producirán con el algoritmo FIFO?

c) (2 puntos – 10 min.) Dada una asignación de 4 marcos de página, y suponiendo que la memoria está inicialmente descargada, ¿cuántos fallos de página se producirán con el algoritmo FIFO?

d) (1 punto – 5 min.) ¿Qué se puede decir de los resultados de **b)** y **c)**?

Pregunta 2 (10 puntos – 50 min.) Raspberry Pi 3 Model B es una computadora de una sola placa de tamaño de una tarjeta de crédito. Su procesador es 1.2GHz 64-bit quad-core ARMv8, la memoria de 1GB, con salida HDMI, 4 puertos USB, Ethernet, WiFi, Bluetooth y MicroSD de almacenamiento. Y su precio es US$ 35. El sistema operativo principal es Raspbian, una versión de Debian, pero trabaja con muchos sistemas operativos tanto basados en Linux como en otros. En MicroSD viene la aplicación NOOBS (New Out of Box Software), la utilidad que facilita la instalación de diferentes sistemas operativos. Inicialmente MicroSD tiene una partición FAT32:

```
$ sudo fdisk /dev/mmcblk0
Disk /dev/mmcblk0: 29,3 GiB, 31444697088 bytes, 61415424 sectors
Sector size (logical/physical): 512 bytes / 512 bytes
Device         Boot Start      End  Sectors  Size Id Type
/dev/mmcblk0p1       8192 61415423 61407232 29,3G  c W95 FAT32 (LBA)
```

Pero con el primer encendido NOOBS instala Raspbian en MicroSD creando 5 particiones:

```
$ sudo fdisk /dev/mmcblk0
Disk /dev/mmcblk0: 29,3 GiB, 31444697088 bytes, 61415424 sectors
Sector size (logical/physical): 512 bytes / 512 bytes
Device         Boot   Start      End  Sectors  Size Id Type
/dev/mmcblk0p1         8192  2289062  2280871  1,1G  e W95 FAT16 (LBA)
/dev/mmcblk0p2      2289063 61415423 59126361 28,2G  5 Extended
/dev/mmcblk0p5      2293760  2359293    65534   32M 83 Linux
/dev/mmcblk0p6      2359296  2488319   129024   63M  c W95 FAT32 (LBA)
/dev/mmcblk0p7      2490368 61415423 58925056 28,1G 83 Linux
```

Ahora nos interesa la partición 5, la más pequeña:

```
$ sudo hexdump -C /dev/mmcblk0p5

00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|   Boot block
*
00000400  00 20 00 00 fc 7f 00 00  66 06 00 00 5f 76 00 00  |. ......f..._v..|   Superblock
00000410  f2 1f 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000420  00 20 00 00 00 20 00 00  00 08 00 00 5d 64 21 58  |. ... ......]d!X|
00000430  5d 64 21 58 05 00 ff ff  53 ef 01 00 01 00 00 00  |]d!X....S.......|
```

Después de **`Boot block`** (bloque #0), siempre con el *offset* `0x400` está el superbloque que tiene el número mágico en `0x438`, e indica (en `0x418`) que el tamaño de bloque es 2^0K = 1024 (`0x400`) bytes (1024 << log_block_size). La partición es de 32MB, o 65534 sectores de 512 bytes, por eso el superbloque indica (en `0x404: fc 7f 00 00`) que el sistema de archivo tiene `0x7ffc` (32764) bloques y (en `0x400`) tiene `0x2000` (8192) *i-nodes*.

Al superbloque sigue el bloque que contiene la tabla de descriptores de grupos de bloques (*block group descriptor table*) (el *offset* `0x800`):

```
00000800  82 00 00 00 86 00 00 00  8a 00 00 00 69 1b f2 07  |............i...|   Group
00000810  02 00 04 00 00 00 00 00  00 00 00 00 f2 07 d9 c8  |................|   descriptors
00000820  83 00 00 00 87 00 00 00  8a 01 00 00 7c 1f 00 08  |............|...|
00000830  00 00 05 00 00 00 00 00  00 00 00 00 00 08 16 d6  |................|
00000840  84 00 00 00 88 00 00 00  8a 02 00 00 00 1c 00 08  |................|
00000850  00 00 05 00 00 00 00 00  00 00 00 00 00 08 2c 1a  |..............,.|
00000860  85 00 00 00 89 00 00 00  8a 03 00 00 7a 1f 00 08  |............z...|
00000870  00 00 05 00 00 00 00 00  00 00 00 00 00 08 0d 78  |...............x|
00000880  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

que describe 4 grupos de bloques (en `0x800`, `0x820`, `0x840`, `0x860`) porque, según el superbloque, cada grupo de bloque es de 8192 bloques (`0x2000` en `0x420`). Las entradas (de 32 bytes cada una) de esta tabla indican que el *blocks bitmap* del grupo 0 está en el bloque # `0x82`, del grupo 1 está en el bloque # `0x83`, del grupo 2 está en el bloque # `0x84`, y del grupo 3 está en el bloque # `0x85`. El *inodes bitmap* del grupo 0 está en el bloque # `0x86`, del grupo 1 está en el bloque # `0x87`, del grupo 2 está en el bloque # `0x88`, y del grupo 3 está en el bloque # `0x89`. El *inodes table* del grupo 0 está en el bloque # `0x8a`, del grupo 1 está en el bloque # `0x18a`, del grupo 2 está en el bloque # `0x28a`, y del grupo 3 está en el bloque # `0x38a`.

Esto es el *blocks bitmap* del grupo 0:

```
000...00  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
000...90  ff ff 7f 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000...a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

(a) (3 puntos – 15 minutos) ¿Cuál es el *offset* es este *bitmap*? (Realice el cálculo en hexadecimal sin calculadora.)

(b) (3 puntos – 15 minutos) Si el grupo 0 es de 8192 (`0x2000`) bloques, ¿cuántos bloques de este grupo están libres según su *bitmap*? (Realice el cálculo en hexadecimal sin calculadora.) Este valor está indicado en el descriptor del grupo.

Esto es el *dump* de *inodes table* del grupo 0:

```
000...00  00 00 00 00 00 00 00 00  05 00 00 00 05 00 00 00  |................|
000...10  05 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000...20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000...80  ed 41 00 00 00 04 00 00  5d 64 21 58 df 02 00 00  |.A......]d!X....|
000...90  df 02 00 00 00 00 00 00  00 00 03 00 02 00 00 00  |................|
000...a0  00 00 08 00 07 00 00 00  0a f3 01 00 04 00 00 00  |................|
000...b0  00 00 00 00 00 00 00 00  01 00 00 00 8a 04 00 00  |................|
000...c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

(c) (2 puntos – 10 minutos) ¿Cuál es el *offset* de esta tabla de *inodes*? (Realice el cálculo en hexadecimal sin calculadora.)

El *inode* #1 es *bad blocks inode* por eso lo saltamos (y cada *inode* es de 128 (`0x80`) bytes) llegando a *inode* #2 que es del directorio raíz. Naturalmente, el *inode* indica en qué bloques se guardan los datos del archivo, y podremos ver sus datos:

```
00122800  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00122810  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00122820  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0d 00 00 00  |lost+found......|
00122830  18 00 0a 01 6e 6f 6f 62  73 2e 63 6f 6e 66 54 4a  |....noobs.confTJ|
00122840  37 36 00 00 0c 00 00 00  1c 00 13 01 77 70 61 5f  |76..........wpa_|
00122850  73 75 70 70 6c 69 63 61  6e 74 2e 63 6f 6e 66 00  |supplicant.conf.|
00122860  0e 00 00 00 a0 03 11 01  69 6e 73 74 61 6c 6c 65  |........installe|
00122870  64 5f 6f 73 2e 6a 73 6f  6e 00 00 00 00 00 00 00  |d_os.json.......|
00122880  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

(d) (2 puntos – 10 minutos) ¿Cuál es el número del bloque que contiene los datos del archivo? ¿Con qué *offset* este número está guardado en el *inode* de este archivo? (Realice el cálculo en hexadecimal sin calculadora.)

<u>Pregunta 3</u> (3 puntos – 15 min.) *(Filesystems – FAT Space Allocation)* El sistema de archivos FAT32 permite, a diferencia que FAT12 y FAT16, seleccionar el tamaño de *cluster* o bloques contiguos de asignación, como 1024 B, 2048 B, 4096 B (default), 8192 B, 16 KiB, y 32 KiB, para volúmenes (o particiones) como los que se tienen en memorias flash USB de 1 GiB, 2 GiB, 4 GiB, 8GiB, 16GiB, etc.

a) (1,5 puntos – 7,5 minutos) Si el usuario tiene en promedio archivos de 3 KiB, ¿cuánto sería el desperdicio, en porcentaje, por fragmentación interna para *clusters* de tamaño 1024 B, 4096 B, 8192 B? ¿Cuánto significa esto en una memoria de 4 GiB, en una de 8 GiB y en una de 16 GiB?

b) (1,5 puntos – 7,5 minutos) Considere un sistema FAT32, con default clúster de 4 KiB, recién creado, siendo el primer *cluster* libre el #3. Se crean archivos de tamaño A de 3 KiB, B de 6 KiB y C de 9KiB. Luego se borra el archivo B de 6 KiB y se crea uno nuevo de D de12 KiB. Muestre la secuencia de ocupación de *clusters* en la tabla FAT, a medida que se crean y borran.

La práctica ha sido preparada por FS (1,3) y VK (2)
en Linux Mint 18 Sarah con LibreOffice Writer.

Profesor del curso: (0781) V. Khlebnikov
(0782) F. Solari A.

Pando, 11 de noviembre de 2016