PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
Examen 2
(Segundo semestre de 2019)

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

Duración: 3 horas
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
El examen debe ser desarrollado en el cuadernillo usando lapicero.
Lo escrito con lápiz será considerado como borrador y NO será evaluado.
Puntaje total: 20 puntos

Pregunta 1 (5 puntos – 45 min.) *(Tanenbaum, A. and Woodhull, A – Minix 2.0.x – OSDI-2E)* Minix 2.0.x se ejecuta en modo protegido de 32 bits, pero sobre memoria real, es decir que las direcciones de memoria corresponden a las direcciones físicas, en un espacio de direccionamiento de 32 bits, con un valor base y un valor límite. Se utiliza una unidad abstracta, *clicks,* para referirse a estas direcciones y como unidad de asignación de memoria. En otros archivos del código fuente, esto se define como 1024, 2048 o similar.

La implementación del asignador de memoria se realiza en una función *alloc_mem(size),* que devuelve la dirección de inicio, en *clicks*, del espacio apropiado de tamaño *size* encontrado, también expresado en *clicks.* El algoritmo usado es *first_fit* o primer ajuste, y estando almacenada la lista de espacios libres en una lista simplemente enlazada, apuntada por un puntero *hole_head,* simplemente se recorre la lista siguiendo el algoritmo, asignando por la dirección más baja, reduciendo el tamaño del espacio libre encontrado o asignando en su totalidad si coincide el pedido con el tamaño del espacio, actualizando así el elemento de la lista en cuanto a la nueva dirección de inicio, y el nuevo tamaño restante, o eliminando el elemento de la lista si hubo coincidencia de tamaño y se asignó en su totalidad (no se mantienen espacios libres de tamaño cero).

Para facilitar la implementación de otros algoritmos, como *Best-fit o Worst-fit,* se ve por conveniente modificar la estructura de los elementos de la lista, de manera que se pueda recorrer tanto por el orden de direcciones, como por tamaño descendente.

Considere que la memoria inicial del sistema, luego de inicializado el sistema y lanzados los primeros procesos de usuario, tiene 4 espacios no contiguos descritos por (inicio, tamaño): (2, 384), (16, 800), (42, 4000), (64, 8000).

a) (2 puntos – 18 min.) Represente la memoria con elementos de una lista ordenada por las direcciones **(0,5 puntos)**. Con el elemento modificado en su estructura, como se describió, represente la memoria en la(s) nueva(s) lista(s) **(1 punto)**. ¿Qué se necesita ahora para acceder en la nueva forma a la nueva lista? **(0,5 puntos)**

b) (2 puntos – 18 min.) Muestre lo que ocurriría con solicitudes sucesivas de memoria, digamos de dos procesos A de tamaño 400 y B de tamaño 1200, luego la terminación de A, y finalmente la terminación de B. Utilice el algoritmo **First-fit (1 punto)** y luego con el algoritmo **Worst-fit (1 punto)**. Puede ayudarse con algún esquema de la situación de memoria en cada momento: antes de las solicitudes, luego de ocurrir ambas solicitudes, y la liberación en igual secuencia, es decir primero se libera lo solicitado por A y finalmente lo solicitado por B.

c) (1 punto – 9 min.) Para implementar **Best-fit**, ¿debería modificarse otra vez la estructura de la lista? En caso afirmativo, justifique. En caso negativo, ¿cuál sería la alternativa?

Pregunta 2 (5 puntos – 45 min.)

a) (2 puntos – 18 min.) (*https://bt.nitk.ac.in/c/17a/co460/*) 48 bit Virtual Addresses, 40 bit Physical Addresses. Page size = 16KB. How many entries in a process's Page Translation Table? What is the size of the Page Translation Table? What is the size of the Inverted Page Table?

b) (3 puntos – 27 min.) (*Thomas Anderson, Michael Dahlin. Operating Systems. Principles and Practice, 2nd Ed.*) Complete las siguientes tablas:

LRU

Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+								
2		B			+										
3				C											
4						D		+							

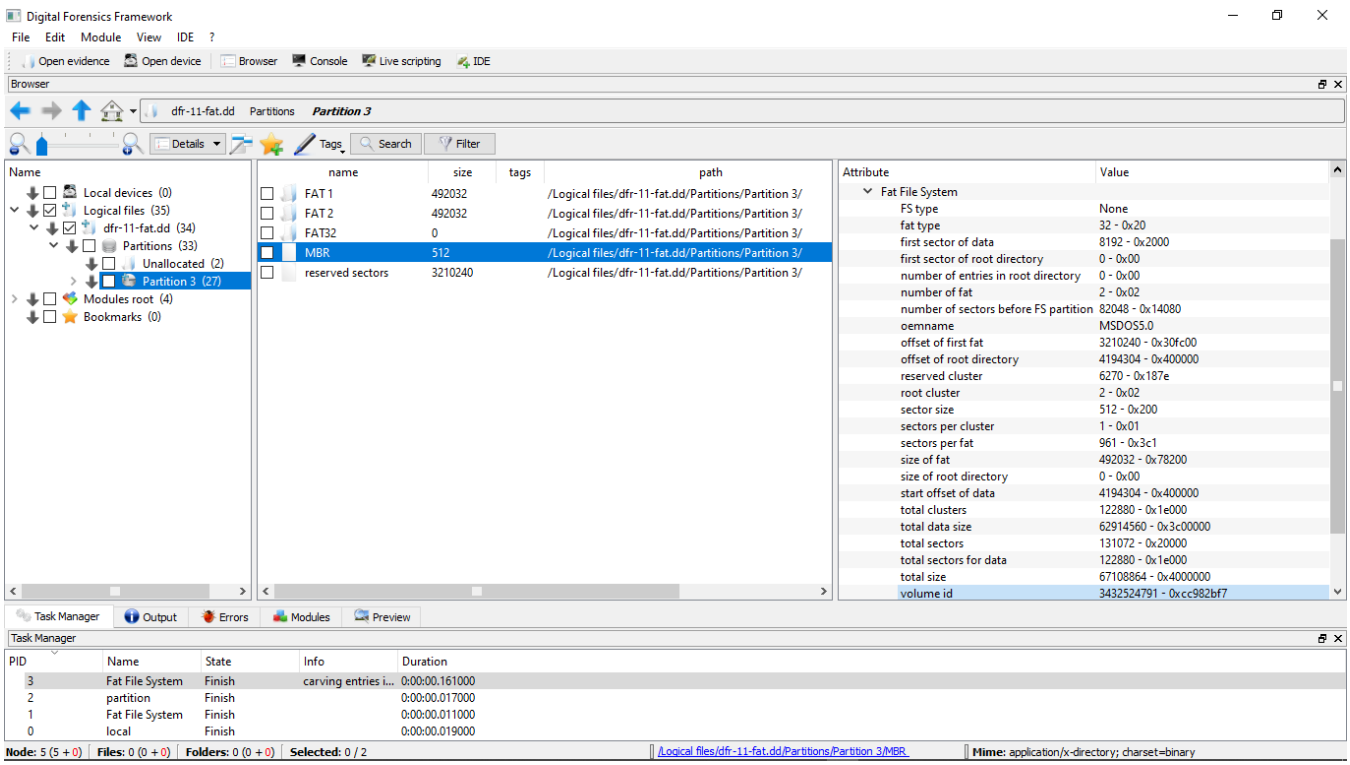
FIFO

Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+								
2		B			+										
3				C											
4						D		+							

OPT

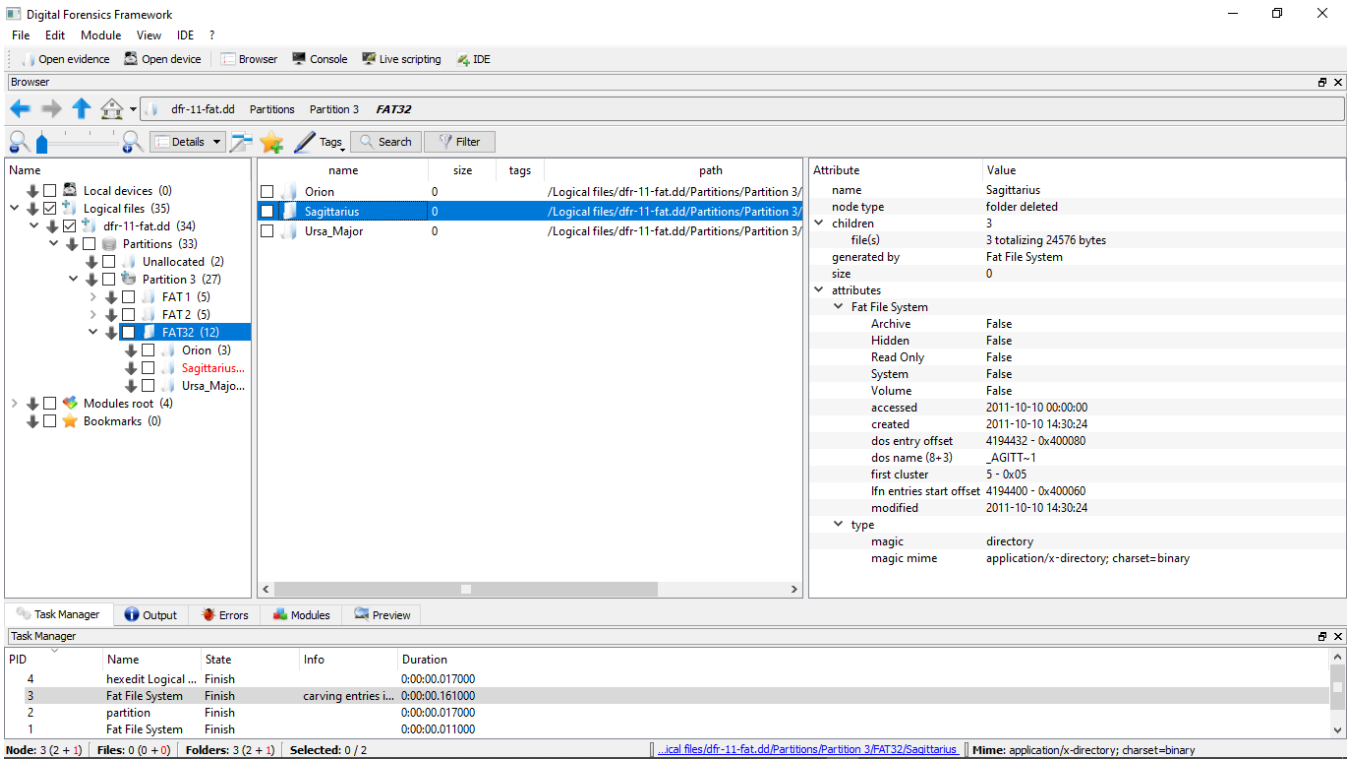
Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+								
2		B			+										
3				C											
4						D		+							

<u>Pregunta 3</u> (5 puntos – 45 min.) Con una herramienta de análisis forense de *filesystems*, de una imagen cruda y directa obtenida de un medio (archivo `dfr-11-fat.dd`) se obtienen los datos que se muestran, para la partición 3:



a) (2 puntos – 18 min.) Determine el sector de inicio de la tabla FAT32, el tamaño de cada una y el total de las dos copias de esta tabla (en sectores), el primer sector del área de datos, que corresponde al *cluster* 2, y la cantidad total de sectores del dispositivo, aproximando a alguna unidad como kilobytes, megabytes, o gigabytes, haciendo un esquema del *LAYOUT* del filesystem FAT32, de inicio a fin, y comprobando el sector que corresponde al *cluster* 2, como primer sector del área de datos.

b) (3 puntos – 27 min.) Del directorio raíz, se obtiene igualmente, la información siguiente de uno de los directorios. Determine: ¿cómo puede intuirse el nombre completo del archivo/directorio borrado? **(0,5 puntos).** ¿Qué *cluster* debería estar "libre" como para recuperar la información? **(0,5 puntos).** Si los directorios, `Orion`, `Sagittarius` y `Ursa_Major`, se crearon en orden, ¿qué *clusters* ocuparán? **(0,5 puntos).** Cada directorio tiene 3 archivos, de 8192 bytes, también creados sucesivamente, ¿qué *clusters* ocupan? **(1,5 puntos).**



<u>Pregunta 4</u> (5 puntos – 45 min.) (*T. Anderson, M. Dahlin. Operating Systems. Principles and Practice, 2nd Ed.*)

a) (2 puntos – 18 min.) "The UNIX Fast File System (FFS) was released in the mid 1980s, and it retained many of the data structures in Ritchie and Thompson's original Unix file system from the early 1970s. In Linux, the popular ext2 and ext3 file systems are based on the FFS design. Suppose BigFS is a variation of FFS that includes in each inode 12 direct, 1 indirect, 1 double indirect, 1 triple indirect, and 1 *quadruple indirect* pointers. Assuming 4 KB blocks and 8-byte pointers, what is the maximum file size this index structure can support?" (Use los prefijos correspondientes: K, M, G, …)

b) (3 puntos – 27 min.) "Tree-based index structures like FFS's can support *sparse files* in which one or more ranges of empty space are surrounded by file data. The ranges of empty space consume no disk space." Por ejemplo,

```
$ cat mksparsefile.c
#include  <fcntl.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <unistd.h>

char    buf1[] = "PUCP";
char    buf2[] = "INGINF";

int
main(void)
{
    int fd;
    off_t to1G = 1024*1024*1024;

    if ((fd = creat("sparse_file.1",S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH)) < 0) {
        perror("creat error");
        exit(1);
    }
    if (write(fd, buf1, 4) != 4) {
        perror("write 1 error");
        exit(2);
    }
    /* offset now = 4 */
    if (lseek(fd, to1G, SEEK_SET) == -1) { /* from the beginning of the file */
        perror("lseek error");
        exit(3);
    }
    /* offset now = 1GB */    /* el espacio vacío se llena con ceros */
    if (write(fd, buf2, 6) != 6) {
        perror("write 2 error");
        exit(4);
    }
    exit(0);
}
```

```
$ gcc mksparsefile.c -o mksparsefile

$ ./mksparsefile

$ ls -lgG sparse_file.1
-rw-r--r-- 1 1073741830 dic  3 13:17 sparse_file.1    # 1073741824 + 6 (1G + 6)

$ ls -lgGh sparse_file.1
-rw-r--r-- 1 1,1G dic  3 13:17 sparse_file.1

$ du -h sparse_file.1    # file space usage on disk in human readable format
8,0K    sparse_file.1    # on ext4 filesystem with blocks of 4 KB (2 blocks)

$ cp sparse_file.1 /media/vk/PUCP_ext2/

$ du -h /media/vk/PUCP_ext2/sparse_file.1
16K     /media/vk/PUCP_ext2/sparse_file.1    # on ext2 filesystem with blocks of 4 KB (4 blocks)
```

El archivo creado tiene la longitud de 1 073 741 830 bytes que deberían ocupar 1 073 741 830 / 4096 = 262 145 bloques de 4 KB, pero siendo un archivo *sparse* (disperso), él ocupa solamente 2 bloques en el sistema de archivos ext4. Los bloques llenos completamente con ceros no se graban al disco. Tampoco estos bloques se graban al disco en el sistema de archivos ext2. Pero, al copiar el archivo creado a este sistema de archivos, él ocupa 4 bloques. ¿Cuál de las diferencias entre los sistemas de archivos ext2 y ext4 se revela en este caso? ¿Cuáles son los 2 bloques adicionales en ext2? Justifíquelo.

Preparado por FS(1,3) y VK(2,4) con LibreOffice Writer
en Linux Mint 19.2 “Tina”

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

Pando, 4 de diciembre de 2019