

**Auxiliar 8 – CC41B**  
**Prof.: Luis Mateu   Aux.: Juan Manuel Barrios**  
**16 de noviembre de 2007**

**Examen 2005, Pregunta 3, parte b.**

Indique cuanto es el espacio perdido por *fragmentación interna* para archivos de: (i) 512 bytes en una partición FAT16 de 256MB, (ii) 512 bytes en una partición FAT16 de 1GB, (iii) 8 KB en una partición FAT32 de 2GB, (iv) 8 KB en una partición Unix de 1GB.

**Examen 2001, Pregunta 3, parte a y b.**

a.- Un programa lee el último carácter de un archivo de 1 GB. El archivo se encuentra en una partición Unix con bloques de 1 KB. Esto se logra en C invocando el procedimiento `fseek` que permite posicionar el cursor de lectura directamente en el desplazamiento  $1\text{GB} - 1$ . Explique cuantos bloques se deben leer de disco (como máximo) para obtener ese carácter.

b.- Suponga que el mismo programa de la parte a se ejecuta 2 veces seguidas. Explique por qué en la segunda ejecución no se realiza ningún acceso a disco.

**Examen 2006, Pregunta 3.**

i. Se tiene un archivo que no requiere bloques de indirección doble en una partición Unix con bloques de 2 KB. Se agrega un byte a este archivo y se crea el bloque de indirección doble. Haga un diagrama mostrando inodo, bloques de datos y de indirección. ¿De qué tamaño es el archivo?

ii. Un programador usuario de un computador Linux descubre que la implementación de `read(fd, n, buf)` en el núcleo no valida que `buf` apunte a un área de memoria del proceso. Explique cómo este programador podría lograr correr su propio código en modo sistema (¡sin ser el usuario root!).

iii. Suponga que un proceso lee el bloque 14. Mientras se lee este bloque en el disco distintos procesos solicitan leer los siguientes bloques de un disco: 15, 13, 18, 7. En qué orden se leerán estos 4 bloques si la estrategia de scheduling de disco es (A) shortest seek first y (B) método del ascensor. Explique.

**Examen 2005, Pregunta 2, parte a y b.**

El siguiente algoritmo ordena ascendentemente un arreglo de enteros usando el método por inserción:

```
int i, j, n= ..., *a= ...;
for (i=0; i<n; i++) {
    int kmin= i;
    for (j= i+1; j<n; j++)
        if (a[kmin]>a[j]) kmin= j;
    swap(&a[i], &a[kmin]);
}
```

a) El algoritmo se ejecuta en una computador con 8 MB de memoria real, el sistema operativo implementa *paginamiento en demanda* con páginas de 4KB, el tamaño del arreglo es de unos 16 MB (i.e.  $n=4$  millones de elementos) y la memoria ocupada por el programa y el sistema operativo es marginal. Haga una *estimación* del número de *pagefaults* al ejecutar este programa.

b) Suponga que el tamaño del arreglo es de 128 KB (i.e. 32 mil elementos) y la TLB (*translation lookaside buffer*) de la MMU (*memory management unit*) posee 16 entradas. Haga una estimación del número de desaciertos en la TLB al ejecutar el programa.

## Solución

### Examen 2005, Pregunta 3, parte b.

(i) Con FAT16 podremos tener  $2^{16}$  bloques direccionables. Como la partición es de 256 MB el tamaño de bloque debe ser al menos de:

$$\begin{aligned}\text{Tamaño Bloque} * 2^{16} &= 256 \text{ MB} \\ \text{Tamaño Bloque} * 2^6 * 2^{10} &= 2^8 * 2^{10} \text{ KB} \\ \text{Tamaño Bloque} &= 4 \text{ KB}\end{aligned}$$

Con un archivo de 512 bytes el espacio perdido por fragmentación interna es de  $4\text{KB} - 512 = 4.096 - 512 = 3.584$  bytes

(ii) Aplicamos la misma fórmula:

$$\begin{aligned}\text{Tamaño Bloque} * 2^{16} &= 1 \text{ GB} \\ \text{Tamaño Bloque} * 2^6 * 2^{10} &= 2^{10} * 2^{10} \text{ KB} \\ \text{Tamaño Bloque} &= 16 \text{ KB}\end{aligned}$$

Con un archivo de 512 bytes el espacio perdido por fragmentación interna es de  $16\text{KB} - 512 = 16.384 - 512 = 15.872$  bytes

(iii) Aplicamos la fórmula para FAT32. Sin embargo no se utilizan los 32 bits para direccionar bloques, si no que solo 28 (hay 4 bits reservados).

$$\begin{aligned}\text{Tamaño Bloque} * 2^{28} &= 2 \text{ GB} \\ \text{Tamaño Bloque} * 2^8 * 2^{20} &= 2^{31} \\ \text{Tamaño Bloque} &= 8 \text{ bytes !!}\end{aligned}$$

Por restricción de hardware siempre el tamaño de bloque será de al menos 512 bytes (ese es el tamaño de un sector de disco duro) => Con un archivo de 8 KB la fragmentación interna sería 0.

Cabe señalar que si ocupamos el tamaño mínimo de bloque la tabla FAT de una partición podría ocupar hasta 256MB de espacio (incluso más si pensamos que se mantienen dos tablas FAT por partición). Eso es demasiado como para mantenerla en memoria, por eso es que el tamaño de bloque con FAT32 normalmente es mayor al mínimo posible (varía entre 4KB, 8KB y 16KB) para mantener el tamaño de la tabla en unos 8MB.

(iv) Para una partición Unix de 1GB basta con un tamaño de bloque de 512 bytes (puede contener archivos de hasta 1 GB).

Con un archivo de 8 KB la fragmentación interna sería 0.

Un valor interesante sería que aún cuando la fragmentación interna es 0 habrá un sobre costo de espacio por usar el primer bloque de indirección de 512 bytes extras.

### Examen 2001, Pregunta 3, parte a y b.

a.- Para tamaños de bloque de 1 KB tenemos que cada bloque de indirección podrá contener 256 punteros. Con este valor, los tamaños que contienen cada puntero de indirección son:

Bloques de datos en inodo:  $12 * 1 \text{ KB} = 12 \text{ KB}$

Indirección simple:  $256 * 1 \text{ KB} = 256 \text{ KB}$

Indirección doble:  $256 * 256 \text{ KB} = 64 \text{ MB}$

Indirección triple:  $256 * 64 \text{ MB} = 16 \text{ GB}$

Si tenemos que leer el carácter de la posición 1 GB tendremos que ocupar el bloque de indirección triple => se leerán 3 bloques de punteros para posicionarse en  $1\text{GB} - 1$ , luego se leerá 1 bloque de dato para leer el carácter.

b.- Al leer los bloques de indirección estos se guardan en caché para no tener que leerlos de nuevo.

### Examen 2006, Pregunta 3.

i.- Si el tamaño de bloque es de 2 KB, cada bloque de indirección podrá contener 512 punteros.

Bloques de datos en inodo:  $12 * 2 \text{ KB} = 24 \text{ KB}$

Indirección simple:  $512 * 2 \text{ KB} = 1 \text{ MB}$

El tamaño del archivo es de  $1 \text{ MB} + 24 \text{ KB}$ .

ii.- Hay que recordar que el método read se ejecuta en modo sistema (también esta es la razón de fondo de por qué un file descriptor es solo un número y no una estructura). Como el sistema ve toda la memoria, el método read podría escribir en cualquier parte si es que no chequea que el buffer apunte a un área de la memoria del proceso que lo invoca. Si los datos del archivo a leer corresponden a un binario y la dirección del buffer apunta a un lugar donde se encuentre otro binario que utilice el sistema, podría ser que el sistema termine ejecutando el código que cargó en vez del comando original que esperaba ejecutar.

iii.- Con la estrategia FIFO los bloques serán leídos en el orden que llegaron: 14, 15, 13, 18, 7.

Con estrategia shortest seek first al estar en el 14 hay que decidir por el 15 o 13. Como no podríamos saber cual está mas cerca (ambos están a distancia 1) tendremos 2 opciones posibles:

Opción A: 14, 13, 15, 18, 7.

Opción B: 14, 15, 13, 18, 7.

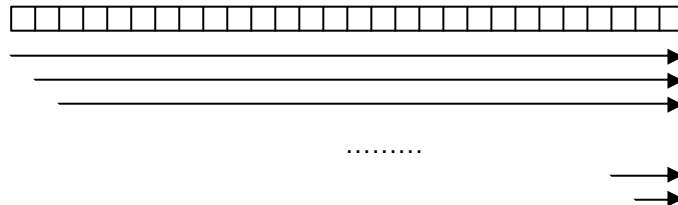
Con estrategia del ascensor hay dos opciones que dependerán de la dirección en que iba cuando leyó el bloque 14.

Opción A (parte subiendo): 14, 15, 18, 13, 7.

Opción B (parte bajando): 14, 13, 7, 15, 18.

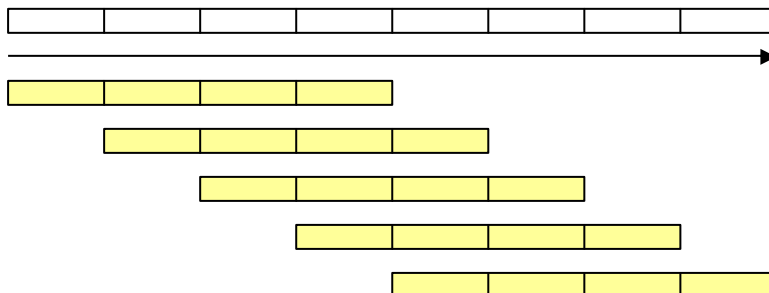
### Examen 2005, Pregunta 2, parte a y b.

a) Vemos que en el algoritmo hace ciclos aumentando en uno el inicio.



Para saber la cantidad de pagefault, primero debemos conocer cuantas páginas utiliza el arreglo. Como el tamaño de página es de 4 KB y el arreglo es de 16 MB, este consta de 4096 páginas (llamemos N a este número). El número de páginas disponibles en la memoria será de  $N/2$ . La cantidad de números por página será de 1024 (llamemos a este número M)

En el primer ciclo, cuando haya cargado las  $N/2$  páginas disponibles y deba leer una nueva página deberá sacar reemplazarla por una ocupada (la que haya referenciado hace más tiempo). Por esto en el primer ciclo hay N pagefaults. El siguiente ejemplo muestra como se moverán las páginas cargadas en memoria:



Cuando comience el nuevo ciclo tendrá que cargar nuevamente la primera página del arreglo porque la que había cargado la reemplazó por otra página. También con la segunda página y las restantes, produciendo nuevamente  $N$  pagefault. Los ciclos comenzarán a iniciarse desde la segunda página una vez que hayan sucedido  $M$  ciclos. Por tanto tenemos  $M \cdot N$  pagefaults, y luego el ciclo se inicia desde la segunda página.

Por el mismo argumento, mientras el ciclo se de inicio desde la segunda página tendremos  $M \cdot (N-1)$  pagefaults.

Estas series de pagefaults se seguirán dando hasta que los ciclos sean tan pequeños que no deba de reemplazar páginas. Esto se dará cuando cruce la mitad del arreglo donde todas las páginas dentro del ciclo estarán siempre en memoria sin producir ningún pagefault.

Tenemos entonces que la cantidad de pagefaults será de  $(N + (N-1) + (N-2) + \dots + (N/2 + 1)) \cdot M$ .

Para calcular hacemos la diferencia entre la sumatoria de 1 a  $N$  con la sumatoria de 1 a  $N/2$ :

$$\frac{M \cdot (N \cdot (N+1) - N/2 \cdot (N/2+1))}{2} \\ M \cdot (3N^2 + 2N) / 8$$

Para dar un número aproximado podríamos quedarnos con el término mayor:  $M \cdot N^2$ . Como  $N$  es  $2^{12}$  y  $M$  es  $2^{10}$  el algoritmo tendrá aproximadamente  $2^{34}$  pagefaults.

b) Ahora como el arreglo es de 128 KB, se compone de 32 páginas y la TLB puede almacenar a lo más 16 entradas. Ahora no ocurrirán pagefaults porque estarán todas en memoria, pero la traducción de pagina virtual a real no será inmediata por el mismo argumento de la parte a. Para hacer una estimación de la tasa de desaciertos usamos la misma aproximación anterior, pero con un  $N=2^5$ , lo que da una tasa de aproximadamente  $2^{20}$  desaciertos de la TLB.