EXAMEN FINAL SO TEMAS 3 Y 4

1. Con respecto a un sistema de paginación resuelva la siguientes cuestiones:

a) Describa detalladamente los pasos que lleva a cabo el núcleo en un sistema de memoria virtual con paginación por demanda cuando se produce una falta de página al tratar de acceder a una dirección de memoria.

Cuando un proceso intenta referenciar una dirección lógica tal que la página en la que está su dirección física traducida no se encuentra cargada en memoria o no es válida (bit de validez 0) se produce una falta de página. En primer lugar, hay que encontrar dónde está la página que se quiere traer a MP de disco y para ello la buscamos en la TUD, luego se busca un marco libre y si no se encuentra ninguno disponible se procede a la aplicación de un algoritmo de reemplazo de páginas. Una vez encontrado se procede a cargar la página, mientras tanto el proceso se pasa a estado bloqueado y una vez cargada en MP se actualiza la tabla de páginas (bit de validez, dirección del marco...) y el proceso cambia su estado a listo para que el planificador lo vuelva a ejecutar reiniciando la instrucción que generó la falta de página.

b) ¿Cuáles son los campos de información (y para qué sirven) que hay que añadir a un sistema de paginación para que soporte memoria virtual? Considere la RAM y el espacio de intercambio (Partición SWAP).

Un sistema de paginación con memoria virtual contiene los siguientes elementos básicos para su correcto funcionamiento:

- Tabla de páginas: contiene campos de información sobre las páginas de un proceso. Sus campos son: dirección del marco de página (physical frame), bit de válidez (se encuentra en MP), bit de modificación (dirty bit) y protección (permisos de acceso).
- Tabla de ubicación en disco (TUD): contiene campos relacionados con las partes de un proceso que no se encuentran en memoria principal. Sus campos son el dispositivo que soporta el bloque de memoria en el que se encuentran y la dirección del bloque en el que se encuentra una determinada página virtual.
- MMU (Memory Management Unit): es unidad hardware encargada de realizar la traducción de una dirección lógica a una virtual haciendo uso de los elementos previamente mencionados. Destacar el uso de los registros TLB (Translation Lookaside Buffer), los cuales se encargan de mantener la correspondencia entre una dirección lógica y una física.
- También destacar la necesidad de una tabla de marcos de página que almacena información sobre los marcos de página y un Swapper que se encargue de decidir qué procesor sacar y llevar de MP al espacio de intercambio.

2. Con respecto a un sistema de paginación resuelva la siguientes cuestiones:

a) ¿Qué problema resuelve la paginación multinivel en un sistema de paginación y cómo interpreta las direcciones virtuales de la CPU dicho sistema?

Vamos a explicar el problema con un ejemplo. Suponiendo que tenemos un sistema de paginación con memoria virtual de tamaño de página 4096= 2^{12} B, tamaño de direcciones 32 bits, luego para el offset tenemos 12 bits y para el número de página tenemos 20 bites, luego tenemos 2^{20}=1048576 páginas de memoria. Suponiendo que cada campo en la tabla de páginas ocupa solo la dirección 32 bits= 4 B la tabla de páginas ocuparía $4 2^{20} = 4 2^{20}$ B= 4 MB, lo cual es un tamaño excesivo ya que esta se tiene que cargar en memoria principal y el tamaño allí es limitado. La solución a este problema es la paginación multinivel, es decir paginar las páginas para reducir el tamaño de la TP, ya que si dividimos la TP en partes del tamaño de una página, podemos páginarla. En este ejemplo hay 4 MB/4096B=1024 = 2^{10} entradas en la TP, luego necesitamos 10 bits para direccionar una entrada, así ahora una dirección logica de 32 bits tendrá 10 bits para identicar la entrada de la tabla de páginas otros 10 bits para identificar la entrada que es y 12 para el offset.

b) Describa las principales estructuras de datos y sus principales campos que utiliza el núcleo de Linux para gestionar el espacio de direcciones virtual de un proceso y su correspondiente espacio físico. Utilizalas para describir el espacio de direcciones del ejercicio 3.

Para gestionar un espacio de direcciones virtual, Linux asigna a cada proceso un espacio de memoria plano. El proceso sólo tiene permiso para acceder a determinados intervalos de direcciones de memoria (áreas de memoria), deberá por tanto haber una estructura que gestione las áreas de memoria asignadas a cada proceso.

Las estructuras básicas de gestión son:

- **PÁGINA**: unidad básica de gestión de memoria. Sus campos son:

```
struct page{
  long flags;
  atomic_t count;
  struct address_space *mapping;
  void *virtual;
  ...
}
```

- **Gfp**: especifica el tipo de memoria que se solicita, pues en Linux hay tres zonas de memoria según la tarea que se quiera realizar.
- **DESCRIPTOR DE MEMORIA**: representa el espacio de direcciones del proceso.

```
struct mm_struct{


  //SECCIÓN 1
  struct vm_area struct *mmap; /*Lista de VMA*/
  struct rb_root mm_rb; /*arbol de VMA para búsqueda*/
  struct list_head mmlist; /*Lista con los espacios
  de direcciones*/


  //SECCIÓN 2
  atomic_t mm_users; /*Procesos usándolo*/
  atomic_t mn_count; /*Contador de referencias*/


  //SECCIÓN 3: todos son unsigned long
  unsigned long start_code;
  unsigned long end_code;
  long start data;
  long end_data;
```

```
  long start_brk;
  long end_brk;
  long arg_start;
  long arg_end;
  long env_start;
  long env_end;


  //SECCIÓN 4: páginas
  pgd_t *pgd;
  long rss;
  long total_vm;

}
```

(Básicamente son 4 secciones dentro del struct)

- **ÁREA DE MEMORIA**: intervalos de direcciones de memoria a los que el proceso tiene permiso para acceder. Describe un intervalo contiguo del espacio de direcciones. Además, contiene un mapa de memoria de la sección de código, de la sección de variables globales inicializadas, de la pila de espacio de usuario y otro con una proyección de la página cero para variables globales no inicializadas.

```
struct vm_area_struct{

  struct mm_struct *vm_mm;

  long vm_start;
  long vm_end;
  long vm_flags;
  struct vm_operations_struct *vm_ops;
  struct vm_area_struct *vm_next; /*lista de VMA*/
  struct rb_node vm_rb; /*El nodo de VMA en el árbol*/
}
```

Ejercicio 3: Considere un sistema con un espacio de direcciones virtuales de 128K (128 * 1024) páginas con un tamaño de 8KB cada una, una memoria física de 64 MB y direccionamiento al nivel de byte. ¿Cuántos bits hay en la dirección virtual? ¿Y en la física? En otras palabras, cuantos bits son necesarios para acceder a una única dirección virtual del espacio virtual y cuantos para acceder a una única dirección física del espacio físico. Direcciones virtuales: 7428, 19425, 22300.

DIRECCIONES VIRTUALES: $128 * 2^{10}K * 8 * 2^{10} = 2^{30}B$, las direcciones virtuales son de 30 bits.

PÁGINAS DE 8KB = 2^{13}B, el offset tiene 13 bits (offset).

De la misma manera, las direcciones físicas tienen 64MB = 2^26 B, tienen 26 bits.

Para traducir las direcciones virtuales 7428 implementamos la tabla de páginas: tenemos 2^30 B y cada página es de 2^13B, por tanto tenemos 2^17 páginas. En este caso suponemos que es contiguo, pero se hace igual en todos.

- Dirección 7428
  $\frac{7428}{8192}$ => resto=7428 cociente=0
  Sabemos que está en la página virtual 0, marco de página 0.

DIRECCIÓN FÍSICA = 7428.

DIRECCIÓN FÍSICA = número de marco*tamaño marco + offset

Ejercicio 4: Con respecto a métodos de asignación de espacio en disco:

a) ¿Qué ventajas presenta el método de asignación FAT con respecto al método indexado con un solo bloque índice?

El inconveniente principal que tiene el método indexado tiene que ver con el espacio en el bloque índice. Si un fichero tiene pocos bloques asignados, dentro del bloque índice se estará malgastando mucho espacio.
En la tabla FAT, sin embargo, se podrá ahorrar espacio, si se establece de antemano el número de bloques que necesita un fichero, para reservar los bloques justos para las entradas de la tabla.

Otra prestación que incluye la tabla FAT es el acceso eficiente (si se encuentra en MP). Simplemente se deberá indexar la tabla por número de bloque, mientras que en el acceso indexado se deberá buscar secuencialmente en el bloque índice hasta encontrar el i-ésimo (metadato).

Por último, la FAT permite un crecimiento de los datos con una implementación relativamente sencilla: bastará con añadir una entrada a la tabla. Sin embargo, para añadir una entrada en un bloque índice que está lleno de manera eficiente, sin causar fragmentación externa, se deben aplicar métodos como la indexación multinivel, que es más difícil de implementar.

b) ¿Qué metodo de asignación de espacio en disco utilizarías para un SA que almacena exclusivamente películas en 4k, y que requiere eliminar peliculas cada 6 meses para almacenar los nuevos títulos que van apareciendo? Justifique.

Para empezar, se necesita un acceso directo eficiente y rápido con archivos grandes y no se debe producir fragmentación externa. Además debe proporcionar sencillez de borrado o inserción, para poder actualizar las películas cada 6 meses. Claramente el método secuencial no resulta adecuado por la poca facilidad de modificación. De la misma manera, el método de asignación enlazado tampoco sería recomendable, pues no permite un acceso directo eficiente.
Las únicas opciones serían la tabla FAT o el método por indexación, observamos que la tabla FAT no resulta adecuada para tratamiento de archivos grandes, pues el tamaño de la tabla crecería de más al añadir películas al sistema.
La asignación por indexación nos presenta el mismo problema que la tabla FAT, entonces la única opción que queda es la indexación multinivel, donde se permite el crecimiento dinámico del sistema, modificaciones sencillas y acceso directo eficiente.

Ejercicio 5: Con respecto al inodo de Unix/Linux:

a) Indica cuáles son los distintos campos que mantiene el inodo y qué información contienen.

El inodo es una estructura asociada a cada directorio y archivo, que mantiene la información sobre él Los campos que puede contener son:

- Identificador del propietario y grupo, UID, GID.
- Tipo de archivo: puede ser regular, directorio, dispositivo de caracteres o bloques, cauce. Si este campo es 0, el inodo está libre.
- Permisos de acceso: dependen del tipo de archivo, o idealmente del usuario.
- Tiempos de acceso: última modificación y acceso.
- Contador de enlaces: almacenan el número de softlinks asociados con el fichero.
- Localización: contenido de las direcciones de los datos en disco del archivo.
- Tamaño.

Cabe decir que un inodo se establece en la Tabla de Inodos, gestionado por el núcleo, que además del contenido del inodo, añade otros campos como por ejemplo:

- **Estado del inodo**, si está bloqueado, modificado, sucio, etc.
- **Número del dispositivo lógico**
- **Número de inodo**
- **Punteros a otros inodos en memoria**
- **Contador de referencias**

b) ¿Qué indica el contador de enlaces del inodo en disco y qué valores puede contener? ¿Qué indica el contador de referencias del inodo de memoria (inodo in core) y qué valores puedes contener?

El contador de enlaces es el número de nombres de ficheros en disco asociados con el inodo, es decir el número de hard links. Si se han borrado todos los enlaces y el contador está a 0, el sistema operativo puede eliminar el archivo del sistema de ficheros. Se puede añadir un enlace (contador+1) si se crea un hard link al fichero, es decir que simplemente se añade otro nombre, las referencias no varían. Por tanto, tiene siempre un valor numérico positivo.

El contador de referencias cuenta cuántas veces el archivo es abierto o accedido, el número de referencias actuales al inodo. Un core-inodo se libera cuando este número es 0, es decir, siempre será un número positivo o nulo. Además, puede resultar útil si se mantiene una lista de inodos libres, pues un inodo pasará a dicha lista sólo cuando su contador de referencias sea 0.

Ejercicio 6: Suponga un inodo de un SA ext3 (Linux) cuyo campo de localización tiene:12 entradas para almacenar números de bloques en disco, una entrada para almacenar el número de un bloque índice, una entrada para almacenar el número de bloque índice que es raíz de una indexación a 2 niveles y una entrada para almacenar el número de bloque índice que es raíz de una indexación a 3 niveles. También suponga que el tamaño de bloque es de 1KB y los números de bloque ocupan 4 bytes:

a) ¿Qué espacio total (en bytes) se requiere para almacenar la información sobre la localización física en archivo que ocupa 3Mbytes? Justifique

Index 1	Index 2	Index 3
12*1024	256*1024	256 * 256 * 1024

Tenemos 1024B/4B = 256 entradas/bloque

El archivo es de 3MB, entonces miramos a ver hasta qué nivel tenemos que ir:

3MB - 12*1024B* - *256*1024B = 2871296 B
Estamos en el tercer nivel de indexación.

$\frac{2871296}{256*1024}$ => cociente = 10; resto=249856

Llega hasta la entrada 10 del primer bloque índice.

$\frac{249856}{1024}$ => cociente = 244; resto = 0

Entonces el espacio gastado en total será:

(12+ 256 + (C1 +1) + C1*256 + C2) * 4B = 12332B

b) ¿Qué bloques índice y qué bloque de datos hay que traer de disco a memoria para acceder al byte 4200000 de un archivo con nombre F1?

Cada bloque índice puede almacenar (1KB / 4B) = 256 entradas de número de bloques.

Se realizan varios accesos a memoria.

Al ser el bloque byte 4200000, no está en el primer nivel ni en el segundo.
Al estar en el tercero, tenemos que calcular de la misma forma el offset en el directorio índice.

4200000 - (12*1024 + 256*1024) = 3925568

3925568 / (256*1024) => c= 14 r = 255552

Está en el bloque índice número 14.

255552 / 1024 => c=249 r= 576

Entonces tendremos que cargar de memoria secundaria el bloque índice del tercer nivel B1, el bloque índice número 14 del B1, llamado B2, y el bloque de datos 576, el que pertenece al byte 4200000.