La implementación de diferentes tipos de memoria en un sistema operativo didáctico

Nicanor Casas ncasas@unlam.edu.ar
Graciela De Luca gdeluca@unlam.edu.ar
Martín Cortina mcortina@unlam.edu.ar
Gerardo Puyo _gpuyo@unlam.edu.ar
Waldo Valiente wvaliente@unlm.edu.ar

Abstract

One of the features of the SODIUM operative system is the variety of memory administrators it has, allowing the student to parametrize the system in accordance with preference and subject to be investigated.
This parametrization covers from the administration of an old mainframe, through contiguous assignations, following with the administration of non contiguous structures, as pagination and pure segmentation, ending up with the structures of demanded pagination and segmentation up to the segmentation/paging in virtual memory.
Let the students "see" the evolution of the administration of resources taking into account the delivery and return of the memory throughout the correspondent tables and from two points of view: the simulation and the execution of processes with different features.

Keywords: Mainframe, Paging, Segmentation, Segmentation/paging, Flat Model, Offset

Resumen

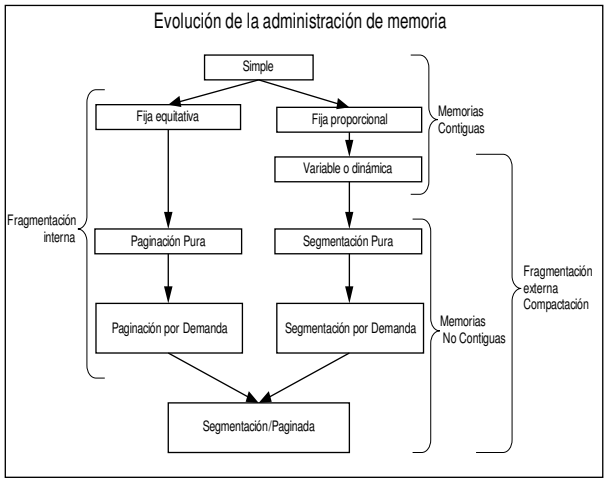
Una de las características que del sistema operativo SODIUM, es la variedad de administradores de memoria que el mismo posee, capaces de permitir al alumno parametrizar al sistema de acuerdo a su conveniencia y al tema que se propone investigar.
Esta parametrización abarca desde la administración de un antiguo mainframe, a través de asignaciones contiguas, siguiendo por la administración de estructuras no contiguas, como ser la paginación y la segmentación pura finalizando con las estructuras de paginación y segmentación por demanda hasta la segmentación/paginada en memoria virtual.
Permitir que el alumno "vea" la evolución de la administración de recursos teniendo en cuenta la entrega y la devolución de memoria a través de las tablas correspondientes y desde dos puntos de vista: desde la simulación y desde la ejecución de procesos de diferentes características.

Palabras Claves: Mainframe, Paginación, Segmentación, Segmentación/paginada, Modelo Plano, Desplazamiento,

1. Introducción

El propósito general que tiene un sistema computarizado es el de ejecutar procesos, ya sean del sistema operativo o del usuario. Los procesos que vamos a ejecutar deben encontrarse en memoria principal total o parcialmente. Cada vez que nos referimos a la administración de la memoria

principal o central, estamos indirectamente hablando de diferentes administradores, gestores o formas de asignarla a los procesos.
Si queremos dar una definición básica de la misma podemos decir que la administración de memoria es la técnica que permite la aplicación de distintos métodos, con sus correspondientes operaciones, que se encargan de obtener la máxima utilidad de la memoria para albergar los procesos que se ejecutan.
La administración de memoria tiene dos partes fundamentales: la concepción o conocimiento de la función por un lado y la generación de la aplicación por el otro.
Desde el punto de vista de la toma de conocimiento, de cara al alumno, explicar cada uno de ellos se convierte en una función confusa, porque en primera instancia parecen estructuras muy diferentes.
Sin embargo explicarnos los administradores de memoria puede ser sumamente fácil debido a que lo único que existe, en definitiva, es el segmento.
Todos parten de un segmento único que tiene un tamaño máximo variable de acuerdo a la posibilidad económica de agregarle memoria. De ahí en adelante el administrador deberá saber si a este segmento lo usará como un solo bloque, como el caso de los sistemas operativos monoprocesos, o lo dividirá en segmentos más pequeños que pueden tener el mismo tamaño o tamaños diferentes.
Para ejemplificar agregamos un cuadro de la evolución de la administración de memoria.



Hasta acá lo enunciado hace simple y fácil de entender a los administradores y parecería que todo termina aquí.
Ahí se terminó la duda y comienzan los diferentes problemas, porque el manejo de segmento tamaños iguales o de tamaños diferentes tiene características puntuales.
Desarrollar en un sistema operativo de características didácticas las diferencias que los caracterizan es una cosa no muy fácil. El alumno debe ver lo que está pasando, debe entender que ventajas y desventajas tiene el uso de uno u otro, y por último debe saber cuales son factibles de aplicar de acuerdo al procesador que va a utilizar.
Veamos ahora de que manera se van sucediendo los diferentes administradores.
Comenzamos con los sistemas operativos que estaban destinados a trabajar con un programa por vez, vemos que la memoria está tomada como un todo ya que la barrera que separa al sistema operativo y del programa puede verse como una línea difusa y no como una frontera entre los dos. Estamos en presencia de un sistema operativo de poco alcance y limitado en sus funciones ya que no aparecen conflictos serios al apropiarse totalmente de la máquina.

Los conflictos aparecen cuando multiplicamos esa forma simple de procesamiento por una característica más avanzada en los sistemas operativos, donde hacen su aparición más de un programa en ejecución al que pasamos a llamar proceso. Aquí aparecen los conflictos de los puntos de carga, de las terminaciones, de las entradas/salidas, de los cambios de contexto, de la sincronización, entre otros, siempre utilizando segmentos de gran tamaño y de forma contigua.
Nótese que hablamos de segmentos pero no decimos si los mismos son fijos o son variables.
Agreguemos ahora que esta diferencia desde el puno de vista del conocimiento de la actividad no es importante porque sólo habría que agregar que el variable requiere reasignar memoria cada vez que sea necesaria, lo que llamamos en el gráfico: compactación.

2. Antecedentes

2.1 Organización de la memoria en IA32

A continuación hacemos referencia a las características que poseen los procesadores que tenemos en nuestros laboratorios y sobre los cuales el grupos de alumnos que cursa la materia sistemas operativos realiza la programación del sistema operativo SODIUM.
La memoria que el procesador direcciona sobre su bus de direcciones se denomina **"Memoria Física",** cada byte es asignado a una dirección denominada **"Dirección Física".** El espacio de direcciones físicas en IA32 posee tres modelos de acceso a memoria:

a) **Modo Real**: que es utilizado por compatibilidad con modelos anteriores (8086), el cual utiliza una memoria segmentada donde las direcciones lógicas están compuestas de un registro que contiene la dirección Base del Segmento, más un Offset, con un tamaño máximo de espacio de direccionamiento de 1 MByte. (***Fig1)***

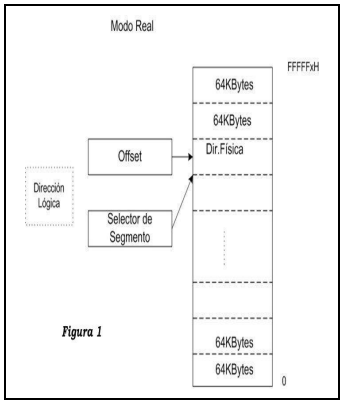


Figura 1

b) Modo protegido

b1) Direccionamiento total: se muestra a los programas como un espacio continuo de direcciones, denominado **Espacio Lineal de Direcciones,** donde los diferentes segmentos están totalmente contenidos en este espacio con direcciones contiguas de 0 a. 2^{32} -1. Esto es conocido por Flat Model de acuerdo a los manuales de INTEL y AMD. ***(Fig2)***

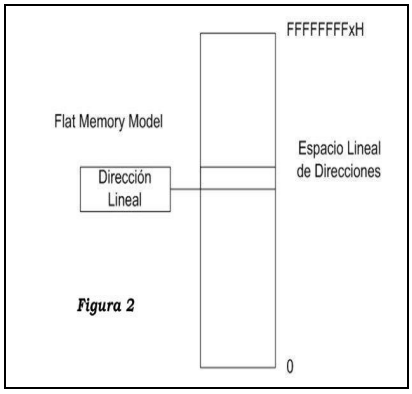


Figura 2

b2) Segmentación: aparece como un grupo de espacios de direcciones independientes: código, datos, stack, y otros, contenidos normalmente en direcciones separadas. Para direccionar un byte en un segmento de programa se usa una dirección lógica que consiste de una Dirección Base de Segmento más Offset. El procesador traduce la dirección lógica en una dirección lineal. ***(Fig3)***

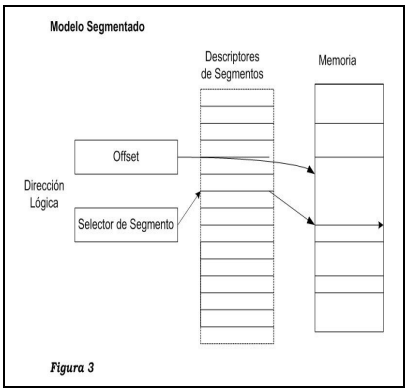


Figura 3

Sobre estos modos de almacenamiento básico el procesador maneja la segmentación y la paginación. La segmentación esta constituida por bloques de memoria de tamaño variable que contienen información de la misma clase y constituye el objeto principal sobre el que se basa el mecanismo de protección. Es eficaz para organizar la memoria en módulos lógicos de similares características que son el soporte de la programación. En modo protegido el procesador necesita dos etapas de traducción de la dirección para llegar a la dirección física y para esto necesita una estructura de datos denominada Global Descriptor Table (GDT), y para algunos procesos una estructura adicional por proceso la Local Descriptor Table (LDT) para la traducción de direcciones. La GDT debe ser creada antes de pasar a modo protegido con su primer descriptor en 0 ya que este no es usado por el procesador. Para acceder a cada descriptor de la GDT o LDT estos son seleccionados a través de registros selectores que indican el numero de entrada en la tabla y un registro del procesador GDTR o LDTR para cada una de las tablas respectivamente los cuales apuntan a la dirección de comienzo de la estructura. Los registros selectores de segmentos ***Fig5*** son registros de 16 bits compuestos por tres campos, el primero de dos bits (0 y 1) se denomina Requested Privilege Level (RPL) y especifica el nivel de privilegio del selector (0 a 3) donde el 0 es el más privilegiado, el segundo es el Table Indicator (TI) utiliza el bit 2, este especifica que tabla utilizar si es 0 utiliza la GDT y si es 1 la LDT. El tercer campo corresponde al Índice de la tabla y

utiliza los bits 3 al 15 del registro, el procesador multiplica el índice por 8 y le suma la dirección del registro GDTR o LDTR obteniendo la dirección del descriptor correspondiente del que obtiene la dirección base del segmento a la cual le suma el Offset y obtiene la dirección lineal. Los registros de segmento son seis de 16 bits en su parte visible pero además tienen una parte oculta denominada shadow register, que se carga con valores del descriptor cuando se hace la primera referencia, para reducir ciclos del procesador en los accesos a memoria. Cada descriptor de segmento es una estructura de datos que ocupa 64 bits con el primer campo que va del bit 0 al 15 contiene los primeros 15 bits del Limite del Segmento, del bit 16 al 31 son los primeros 16 bits del campo Dirección Base, en los siguientes 32 bits se encuentra del 0 al 7 los bits16 al 23 de la Dirección Base, del bit 8 al 11 es el campo de Tipo, el bit 12 es el tipo de descriptor (S) si esta en 0 es una segmento del sistema y si está en 1 es un segmento de código o datos, los bits 13 y 14 corresponden al Descriptor Privilege Level (DPL) que posee un rango de 0 a 3, el bit 15 corresponde al campo Segmento Presente (P) el cual indica si vale 1 que el segmento está presente en memoria, del bit 16 al 19 corresponde a los bits 16 a 19 del limite del segmento, el bit 20 está disponible para software del sistema, el bit 21 si está en 1 indica que está trabajando en modo de 64 bits, el bit 22 Default operation size / default stack pointer size / or upper bound flag (D/B) posee diferentes funciones dependiendo de si el descriptor es de un segmento de código ejecutable, un segmento de datos expand down o un segmento de stack, si el bit esta seteado en 1 es para segmento de código o datos de 32 bits y en 0 para código o datos de 16 bits, el bit 23 corresponde a la Granularidad (G) si el bit está en 0 el límite de los datos está en bytes en cambio si se setea en 1 el limite de los datos es interpretado en unidades de 4 Kbytes, los bits del 24 al 31 indican los bits 24 al 31 de la dirección base del segmento.

Para trabajar en modo protegido necesitamos también la estructura de datos Interrupt Descriptor Table (IDT) donde asocia a una excepción o vector de interrupciones con un Gate descriptor para la tarea utilizada para el servicio de la excepción o interrupción, Task – State Segment (TSS) permite salvar el contenido del procesador de una tarea dada para luego recuperarla, esto nos permite la multitarea, junto con las otras estructuras anteriormente expuestas para el manejo de memoria segmentada en Intel.

La segmentación en los procesadores Intel no puede ser deshabilitada, solo puede ser limitada en cuanto a los controles que ofrece. Por lo tanto la paginación puede implementarse como paginación sin segmentación o como segmentación paginada. La Paginación sin segmentación se puede implementar sobre un Flat Model Protegido ***Fig.4*** o sobre un Flat Model Básico. ***Fig.5*** El modelo básico permite tener un acceso continuo a una memoria no segmentada. Para su implementación es necesario tener por lo menos dos descriptores de segmentos creados, uno para el código y otro para los datos, ambos mapeados en el espacio de direcciones entero.

El Modelo protegido es similar al anterior pero el límite de los segmentos debe ser la memoria existente, donde cualquier acceso a memoria a una dirección inexistente provoca una excepción general de protección por acceso a memoria inexistente. Como vemos en ambos modelos no podemos prescindir de los descriptores de segmentos que utilizamos para traducir las direcciones y acceder a memoria.

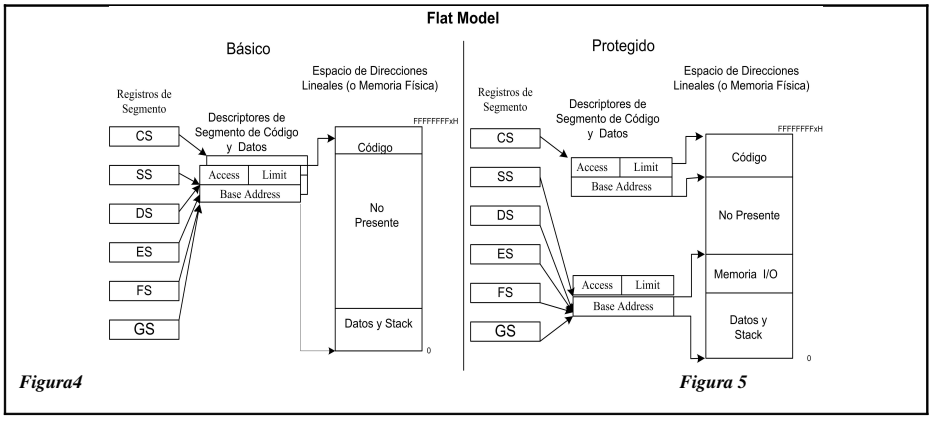


Figura4 *Figura 5*

La Paginación permite que el espacio de direcciones lineales sea mapeado directamente sobre la memoria física o indirectamente sobre la memoria física y almacenamiento en disco, este último se refiere a la paginación por demanda (Memoria Virtual). Cuando se referencia una dirección lógica en memoria el procesador traduce esta a una dirección lineal y entonces utiliza el mecanismo de paginación para traducirla a una dirección física. Los tamaños de página disponibles en este sistema son de 4 Kbytes o 4 Mbytes la dirección lineal que utiliza el mecanismo de paginación es de 32 bits y está compuesta por el número de entrada al directorio de páginas, número de entrada en la tabla de páginas y Offset para páginas de 4 Kbytes, las de 4 Mbytes están compuestas por número de entrada en el directorio de páginas y Offset. ***Fig.6***

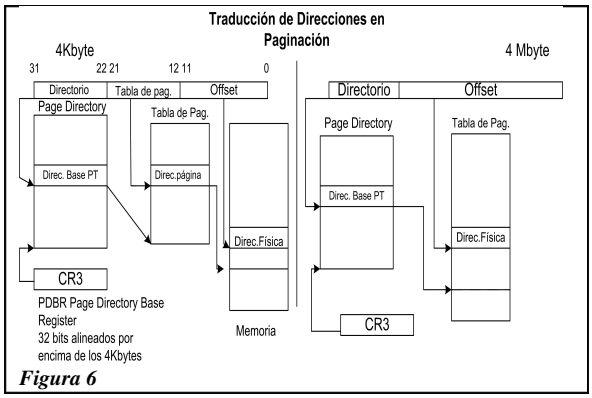


Figura 6

El registro CR3 es utilizado como puntero al directorio de páginas y cada una de las entradas del directorio de páginas contiene la dirección física de la base de la tabla de páginas correspondiente, en el caso de ser utilizada dicha tabla, de no ser así cada entrada del directorio de páginas contiene la dirección física de la página de memoria alineada a 4Mbytes. La paginación es controlada con tres flags : PG (paging) bit 31 de CR0 habilita la paginación, PSE (page size extensión) bit 4 de CR4 (Pentium en adelante) habilita el tamaño de 4kBytes o 4 Mbytes si PAE=0 y 4Mbytes o 2 Mbytes si PAE=1 y PAE (phisical address extensión) Bit 5 de CR4 (Pentium Pro en adelante) habilita la extensión de direcciones a 36 bits. El contenido de las entradas en el directorio varía si se utilizan paginas de 4KBytes o 4MBytes.

2.2.- Informe de System Calls de diversos Sistemas Operativos

Los sistemas operativos emplean distintos tipos de system calls, entre las más conocidas se encuentran POSIX, WIN32, OS2, entre otras.

En el caso de Win32 es utilizada por todos los sistemas operativos de Microsoft desde Windows 9x, Windows NT, Windows 2K y las sucesivas releases. Las interfaces provistas son complicadas, abusan de los sistemas de tipos, causa polución en el espacio de nombres, uso inconsistente de las convenciones de nombres. En resumen la funcionalidad de la interfase va desde la inconsistencia a estados incompletos e inadecuados de documentación, según Diomidis Spinellis, por lo que no la podemos considerar un estándar. Por otro lado POSIX es un estándar aprobado por la IEEE con diversas versiones, en su definición únicamente establece los nombres de las system calls con su número correspondiente, y los parámetros que pasa con su tipo correspondiente y el parámetro de retorno siempre en el registro eax, el cual se mantiene en todas las versiones, dejando a cada sistema operativo libre de desarrollar sus propias rutinas de atención de acuerdo a sus criterios de diseño. La mayoría de los sistemas operativos actuales han optado por este estándar ya que les permite portabilidad entre distintos sistemas operativos y utilizando bibliotecas dinámicas, también tenemos portabilidad en las distintas arquitecturas. El sistema OS2 fue desarrollado por IBM como continuación del DOS para PC pero sus sistemas operativos actuales como AIX utilizan POSIX y la mayoría de sus máquinas trabajan con el sistema operativo Linux que también utiliza con POSIX. OS2 es soportado por Windows NT y 2Kx pero pocos otros sistemas operativos los soportan y posee poca documentación de las interfases.

3.- Los gestores de Memoria en SODIUM

Las zonas de memoria asignadas a un proceso constan de un área para código, datos del proceso, compuestos por: área de datos donde se ubican las variables inicializadas, la bss que contiene variables no inicializadas, el heap y el stack del proceso. El sistema de gestión de memoria debe cumplir funciones esenciales como la contabilización del estado de la memoria, la asignación, liberación y protección. La asignación de memoria se da cuando se crea un proceso o durante su ciclo de vida, cuando necesita incrementar su tamaño y la liberación cuando finaliza o cuando devuelve áreas asignadas durante el ciclo de ejecución.

3.1.- El gestor de memoria para monoprogramación.

El comenzar por este gestor de administrador nos dio un gran impulso para continuar con los otros. Se utilizó completamente la característica de la memoria real y si bien se redujo el tamaño para ser congruente con los procesos que se compilaron bajo lenguaje C.
Se procedió primeramente a generar una simulación que consta de las siguiente posibilidades:

a) Determinar el tamaño máximo de la memoria a utilizar
b) Ingresar un tamaño de programa simulado
c) Determinar la fragmentación externa existente.
d) Determinar la memoria total ocupada entre el sistema operativo y el proceso.

En segundo término se procedió a generar programas y cargarlos en memoria donde se obtuvo.

a) Determinar el tamaño del programa ingresado
b) Determinar la fragmentación externa
c) Determinar la memoria total ocupada entre el sistema operativo y el programa.

3.1.1 Investigaciones pendientes

El equipo de investigación está abocado en la actualidad a generar segmentos de tamaño máximo habilitando la compuerta A20 y provocando direccionamientos de hasta 1 GB sin generar una tabla de descriptores GDT.

3.2.- La memoria para multiprogramación de particiones variables.

Unos de los problemas fue recrear los sistemas de gestión continuos dado que hay que establecer diferentes clases para el gestor de memoria contigua particionada dado que en la actualidad los mainframe no lo usan con frecuencia.

En el caso de asignación contigua se administra el espacio libre u ocupado mediante una lista doblemente enlazada de huecos libres. La asignación se puede hacer mediante los algoritmos First Fit, Next Fit, Best Fit, o Worst Fit, permitiendo ver el estado de asignación de memoria y la fragmentación producida, guardando los distintos estados de la memoria para realizar comparaciones de comportamiento. Como esta administración de memoria se realiza sin deshabilitar la GDT, esto nos permite una protección mínima ya que al proceso a ejecutar se le asignan dos descriptores de segmento uno para los datos y otro para el código, ambos con la misma base que corresponde al punto donde el proceso es cargado y el mismo límite que corresponde al tamaño que este proceso completo ocupa en memoria. Debido a este límite se evitará que cualquier dirección del proceso acceda fuera de sus direcciones asignadas (espacio de direccionamiento).

3.2.1 Investigaciones pendientes

Asignar la memoria bajo el sistema de BUDDY SYSTEM verificando las variaciones en la fragmentación utilizando sistemas de compactación y de memoria total disponible con tamaños no potencia de dos.

3.3.- Gestor de segmentación.

Otra forma de administración implementada es la segmentación, donde también se asigna la memoria mediante una lista doblemente enlazada, con los algoritmos anteriormente expuestos y con las mismas funciones para ver el estado de asignación de la memoria y la fragmentación producida. Como hemos dicho anteriormente este sistema es igual que el de particiones contiguas variables pero con la diferencia de estar compuesto cada proceso por segmentos más pequeños.

En este caso como a los segmentos de código, datos, heap y stack se le asignan diferentes descriptores, cuya dirección base corresponde al punto de carga de cada uno de estos en memoria, esto permite agregar protecciones de ejecución, lectura y escritura a los segmentos correspondientes además de tener límites para impedir el direccionamiento fuera del área asignada. Para los procesos la arquitectura nos permite utilizar la tabla global GDT con dos descriptores o una LDT por proceso con todas las entradas necesarias para los diferentes segmentos.

En los dos casos antes mencionados la lista enlazada de huecos libres de memoria va generando nuevos nodos en la medida en que los procesos finalizan o devuelven memoria, con la condición de unir dos nodos y generar uno con más espacio disponible y las áreas de memoria liberadas son contiguas.

Esto nos permitió verificar que las asignaciones de espacio libres (huecos) a través de listas enlazadas se transforman en ciclos lo que es poco beneficioso para la asignación de memoria y la devolución.

3.3.1 Investigaciones pendientes.

Para la asignación de memoria se realizará la contabilización de asignaciones ocupadas y libres generando nodos de procesos o huecos respectivamente en una lista enlazada, lo que nos permitiría tener el estado total de la memoria y se deberá verificar si es más eficiente la asignación o más

lenta, si esto mejora con un enlace adicional de nodos de huecos libres y que costo esto tendría, en los tiempos de búsqueda y de generación de la lista, esta implementación haría que cuando se liberan procesos contiguos a los huecos, automáticamente se genere un hueco más grande.

3.4.- Gestor de paginación

La tercera forma de administración es la paginación, para la cual la contabilización del espacio libre se maneja con un bit-vector de frames libres/ocupados (0 libre, 1 ocupado), para la traducción de direcciones lógicas en direcciones físicas utilizamos las estructuras de datos, donde las direcciones lineales corresponden a una paginación en dos niveles con un tamaño de página de 4Kbyte. Los bits 0 al 11 corresponden al Offset, los bits 12 al 21 corresponden al número de la entrada en la tabla de páginas y los bits 22 al 31 corresponden al número de entrada al directorio de páginas. No deshabilitamos la GDT pero la minimizamos ya que para trabajar en paginación utilizamos dos entradas en dicha tabla con dirección base igual a cero y límite igual al tamaño que el proceso paginado ocupa en memoria, es decir un múltiplo de 4Kbytes. Esto hace que la dirección lógica que genera el compilador sea exactamente igual a la dirección lineal que va a utilizar la MMU para la traducción de direcciones lineales a direcciones físicas de memoria. Como el tamaño de la página es potencia de 2 el MMU puede separar directamente el Offset y quedarse con los bits restantes, de los cuales los siguientes a separar serán los correspondientes al número de página y los últimos corresponderán al número de entrada en el directorio. Esta división en dos niveles la realizamos debido a que cada tabla de páginas tiene 1024 entradas de 32 bits cada una con la dirección del punto de carga alineada a 4k que nos permite utilizar los primeros doce bits para administración y control de la manera más conveniente para nosotros. Estas entradas pueden estar utilizadas o no, para esto la entrada en la tabla tiene un bit de validez que vale uno para las páginas existentes y cero para las otras. Cuando se referencia a una página no válida este direccionamiento genera una excepción de dirección inválida, que nos permite tratarlo mediante una rutina que muestra el error y dispara la finalización del proceso. La paginación en dos niveles, nos permite ocupar menos espacio en memoria con las tablas de páginas ya que de no ser así tendríamos una tabla por proceso de 2^{20} entradas (con tamaño de 4Mbytes), en cambio de esta manera podemos tener para un proceso chico únicamente dos tablas una del directorio de páginas (que siempre es única) y la otra que es la tabla de páginas que contiene los puntos de carga de los procesos en memoria (cada una ocupa 4Kbytes). Aquí también utilizamos los algoritmos First Fit y Next Fit para las asignaciones mostrando la sencillez de la asignación y recuperación de memoria.
También contabilizamos fácilmente la fragmentación interna producida cuando cargamos los procesos, permitiéndonos las comparaciones con otros modelos de administración de memoria. La desventaja de la paginación es el aumento de la complejidad y los tiempos de la traducción de direcciones. Para disminuir estos tiempos se cuenta con un buffer en el procesador (TLB) que contiene las últimas direcciones referenciadas por el proceso que está en ejecución, el cual solo puede limpiar el sistema operativo, cuando cambia el contenido del registro que apunta al directorio de páginas, el cual modificamos en cada process switch.

3.4.1 Investigaciones pendientes.

Implementar técnicas de paginación que combinen diferentes tamaños de páginas para programas con tamaños muy diversos. Sabemos que las pruebas de escritorio llevadas a cabo determinan la aparición de fragmentación externa en el sistema.

4.- System calls para memoria

Cuando los procesos están en ejecución hacen solicitudes dinámicamente, estas las hacen a través de los system calls.
POSIX posee una serie de llamadas para realizar los servicios, estas son brk, calloc, free, malloc, mlock, mlockall, mprotect, munlock, munlockall, realock y sbrk. Estás son la que estrictamente corresponden a la gestión de memoria.

4.1.- System calls en SODIUM

Nuestro sistema implementa únicamente los sytem calls para asignación y devolución de memoria que corresponden a las funciones brk y sbrk del estándar POSIX debido a que el resto de los nombrados anteriormente deben ser tratados a través de las bibliotecas que manejan las APIS.

4.1.1.- Cambio de tamaño de la asignación del segmento de datos.

Un proceso puede modificar el tamaño de datos, para ello se utiliza la llamada al sistema **brk**, cuya sintaxis es la siguiente int brk(void *end_data_segment), donde el parámetro end_data_segment especifica la dirección final del segmento de datos, debe ser superior a la dirección del segmento de código e inferior en 16 Kbytes a la dirección del fin de segmento de pila.
La otra llamada que le permite modificar el tamaño del segmento de datos es **sbrk** y la sintaxis correspondiente es void *sbrk (ptrdiff_t increment) el parámetro increment indica el número de bytes a añadir o quitar del segmento, si es positivo o negativo respectivamente. Ambas cuando fallan devuelven -1.

4.1.2.- APIS de asignación y liberación de memoria

La system call malloc, sirve para poder asignar espacio en memoria dinámicamente. Esta brinda la posibilidad de poder utilizar una mayor cantidad de memoria de la que haya solicitado el proceso inicialmente. El programa al compilarse, linkeditarse y generar el programa ejecutable, deja en el comienzo del ejecutable el espacio de memoria necesario para ejecutarse. El valor inicial es sólo para reservar la memoria estática del proceso, esto significa que reserva el espacio para todas las variables y el código del programa. Estas áreas corresponden a los segmentos de código y de datos. Externo a dichos sectores se encuentran el área de stack y el área del heap (esta última corresponde a la memoria asignada dinámicamente)
Para el caso en el cual el programa utilice punteros, inicialmente solo se les guardará espacio para almacenar el puntero ya que al momento de poner a ejecutar el programa solo se sabe cuanto ocupa el puntero de la memoria dinámica no la cantidad de memoria que se utiliza para guardar la estructura. Este valor de la cantidad de memoria dinámica que se utiliza puede variar en cada ejecución de un mismo programa. Es por eso que siempre se utiliza un área de HEAP por defecto y en caso de que el proceso necesite más, se tendrían que extender los límites del HEAP.

4.1.3¿Cómo funciona?

La función del sistema operativo recibe como parámetro la cantidad de bytes que necesita el proceso. Se busca en el HEAP la cantidad de espacio disponible para la cantidad de bytes solicitados. Si encuentra el espacio necesario lo marca como ocupado y devuelve un puntero a la posición de memoria encontrada.
En caso de no encontrar más espacio en el HEAP del proceso usuario, nuestro sistema operativo tendrá que analizar si hay espacio disponible en el resto de la memoria y hacer crecer el HEAP con

lo cual es extenderá el proceso y tendrá la cantidad de bytes que pidió. Si lo que el sistema nos agrega para poder utilizar el HEAP sigue siendo insuficiente para la cantidad de bytes que envió el proceso usuario, la función devuelve un valor nulo y el mismo cancelará.

4.3.- Beneficios

El mayor beneficio está dado en la implementación de POSIX, ya que existe mucha documentación disponible, código abierto de diversos Sistemas Operativos (como Linux, Minix, Solaris Open Source) que permiten comprender mejor su funcionamiento, la portabilidad, ya que un proceso creado en Linux con la biblioteca adecuada podría ejecutar en nuestro sistema permitiéndonos disponer de una variedad de programas para la realización de pruebas y estadísticas.

4.4.- Investigaciones pendientes

POSIX está diseñado para trabajar con una forma de administración y nuestro sistema operativo durante su ejecución permite cambiar, por lo que debemos tener diferentes rutinas para ejecutar la misma system call pero con el sistema parametrizado de otra manera. La administración de memoria presenta el problema de convertir los segmentos de los procesos en frames de memoria para pasar de memoria segmentada a memoria paginada, pero de memoria paginada pura a memoria segmentada deberíamos analizar el proceso para transformar las páginas, que además incluyen fragmentación interna, en segmentos, necesitando un área de intercambio o hacer vuelco a disco de los procesos en memoria. Para realizar este pasaje de páginas a segmentos, en un momento dado, se deben convertir frames libres/ocupados para páginas al mismo tiempo que huecos libre/ocupados para los segmentos.
La decisión más sencilla que se ha tomado en el presente, ya que ésta no afecta la didáctica del sistema, es eliminar los procesos de usuario, reconfigurar el sistema operativo y volver a arrancar los procesos ya que nuestro objetivo es mostrar las asignaciones de memoria, la traza de ejecución, la reubicación de los procesos y la liberación de memoria tanto para la paginación como para la segmentación.

5.- Conclusión

La ventaja principal que observamos al conservar los datos de los estados de asignación de memoria durante la ejecución de un conjunto de procesos de prueba para cada uno de los gestores, es que permite la comparación de los distintos modelos de administración. La observando en forma gráfica de la ubicación los procesos, desperdicio que se produjo, funcionamiento de los algoritmos de asignación y el estado de las estructuras utilizadas en cada momento hace más comprensible a los alumnos el funcionamiento de la asignación de memoria.
También es interesante ver la traducción de instrucciones de lógicas a físicas, sus diferencias y similitudes, permitiendo ver el contenido de las tablas de directorio y de páginas, así como los descriptores de la GDT o LDT correspondientes a los procesos, junto con sus direcciones antes y después de cada traducción. Uno de los puntos interesantes de la generación de direcciones dinámicas es ver como las direcciones lógicas generadas por el compilador son independientes del tamaño de la página utilizada y del punto de carga en donde va a ser asignada una parte del proceso, como también un número de frame es para el procesador equivalente a una dirección física completa. Esto le permite entender al alumno la razón por la cual los tamaños de páginas deben ser potencias de 2. También mostrando la búsqueda de espacio disponible mediante las listas enlazadas podemos ver la dificultad que presentan los algoritmos de asignación, ya que éstos deben ser recorridos en forma secuencial.

6.- Bibliografía

Angulo J M. & E. M. Funke, *Microprocesadores Avanzados 386 y 486 Introducción al PENTIUM y PENTIUM PRO,* Editorial Paraninfo, Cuarta Edición, 1998

Brey B., *Los Procesadores Intel,* Prentice Hall, Quinta edición, 2000

Deitel H. M *Sistemas Operativos*, Addison Wesley, Segunda edición, 1990

IA-32 *Intel Architecture Software Developer's Manual*, Volume 3: System Programming Guide, 2003.

Milenkovic M. *Sistemas Operativos Conceptos y diseño,* Mc Graw Hill, Segunda edición, 1994

Neetzel C. *Notas sobre Sistemas Operativos* (MANUAL DEL ALUMNO), Rocamora, Buenos Aires, 1998

Pinkert J. & L.Wear Operating Systems: Concepts, Polices and Mechanisms, Englewood Cliffts NJ, Prentice Hall, First Edition, 1989

Standard Microsystems Corporation *Application note 6.12,* 2000

Silverschatz A. & P. Galvin *Operating System Concepts* Addison-Wesley Longman, Fifth Edition,

Stallings W. *Operating Systems Internals and design principles,* Prentice Hall International, Third Edition, 1998

Tanenbaum A.S. & A. S. Woodhull *Operating Systems Design and Implementation,* Prentice Hall, Second Edition, 1997

Tanenbaum A. S. & A. S. Woodhull, *Operating Systems Design and Implementation,* Second Edition

Turley J. *Advanced 386 Programming Techniques,* Editorial Osborne McGraw-Hills, Primera Edición, 2004

Revistas y Publicaciones

Diomidis Spinellis - Computer Standars & Interfaces 20:1-8 November 1998 "A Critique of the Windows Application Programming Interface" University of the Aegean

BAYS, C.: "A Comparison of **Next**-**Fit**, First-**Fit**, and Best-**Fit**", Commun. of the ACM, vol. 20, pp.# 191-192, marzo de 1977

Internet

Anthony J. Massa, Embedded Software Development with eCos

Christopher R. Hertel, Implementing CIFS: The Common Internet File System

Jamie Cameron, Managing Linux Systems with Webmin: System Administration and Module Development

Jasmin Blanchette, Mark Summerfield, C++ GUI Programming with Qt 3

Mel Gorman, Understanding the Linux Virtual Memory Manager

Nigel McFarlane, Rapid Application Development with Mozilla

Rafeeq Ur Rehman, Christopher Paul, The Linux Development Platform: Configuring, Using, and Maintaining a Complete Programming Environment

Rafeeq Ur Rehman, Intrusion Detection Systems with Snort: Advanced IDS Techniques with Snort, Apache, MySQL, PHP, and ACID

Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification Version 1.2 TIS Committee May 1995