Desarrollo de un sistema operativo didáctico

Universidad Nacional de La Matanza - Departamento de Ingeniería e Investigaciones Tecnológicas
Florencio Varela 1603 – (1754) – San Justo – Buenos Aires – Argentina

Ing. Nicanor Casas
ncasas@unlam.edu.ar

Lic. Graciela De Luca
gdeluca@unlam.edu.ar

Sr. Martín Cortina
mcortina@unlam.edu.ar

Abstract

The main purpose of the SODIUM operating system is to allow students to compare different algorithms of processes administration, memory administration, input/output handling devices and diverse file system, using for this a parameterization at execution time. This will permit them to perform execution processes tests selecting different wanted algorithms, avoiding the making up of tedious compilations, or written comparisons, which are normally not especified in common course books because the problems solving is limited by exercises of simple input.

Other additional SODIUM purpose is to work out the algorithms in their original form, in the way they were design by their creators, and following a basic structure, bypassing in this way the characteristic adaptations of the best known operative systems, that compete for the informatic market access.

The SODIUM system also allows the different computer constitutive elements to be virtualized so that the algorithms can work and be consulted in their purest form without depending on other elements that can impair their performance.

Keywords: Kernel, Loader, Unix, FIFO, SJF, Round Robin, Priorities, FAT, FAT, IVT, BDA, DPL, GDT, PIC, TSS, RTC, IDT

Resumen

El propósito principal del sistema operativo SODIUM es el de permitir a los alumnos comparar diferentes algoritmos de administración de procesos, de administración de memoria, de manejo de dispositivos de entrada/salida y diferentes sistemas de archivos, utilizando para ello un sistema de parametrizaciones en tiempo de ejecución. Esto les permitirá realizar pruebas de ejecución de procesos seleccionando diferentes algoritmos a voluntad, evitándose tener que realizar tediosas compilaciones, o tener que realizar comparaciones escritas, que por lo general no se encuentran especificadas en los libros más comunes de la asignatura dado que la resolución de problemas está limitada a ejercicios de simple entrada.

Otro propósito del SODIUM es el de trabajar los algoritmos en su forma original, tal como fueron planteados por sus creadores, y siguiendo una estructura básica, de esta forma evitando las adaptaciones características de los sistemas operativos más conocidos, que compiten por la penetración en el mercado informático.

El SODIUM posibilita a su vez que los diferentes elementos constitutivos de la computadora puedan ser virtualizados a fin de que los algoritmos puedan funcionar y ser consultados en su forma más pura sin tener que depender de otros elementos que puedan deteriorar su rendimiento.

Palabras Claves: Núcleo, Cargador, Unix, FIFO, SJF, Round Robin, Prioridades, FAT, IVT, BDA, DPL, GDT, PIC, TSS, RTC, IDT

1. Antecedentes

Podemos dividir los antecedentes en dos tipos diferentes de actitudes que se adoptaron para la realización de este sistema operativo.

La primera fue la del estudio de los sistemas operativos actuales más comunes en el ámbito universitario como ser Windows y Linux.

La segunda es la búsqueda de otros sistemas operativos de características didácticas desarrollados en otras universidades.

1.1.- Área Windows y Linux

Con respecto a la primera el sistema operativo Windows en cualquiera de sus versiones no está abierto a los usuarios generales ni a las universidades por lo que realizar modificaciones al respecto es muy complejo y de difícil implementación para una materia que corresponde a la currícula general del plan de estudios.

En lo que respecta a Linux teníamos la ventaja de tener a nuestro alcance el código abierto del mismo y quedaba la opción de partir de un Kernel estable y probado y a partir de allí realizar las modificaciones necesarias para poder adaptarlo a nuestros requerimientos.

Después de un período importante de tiempo dedicado al estudio del mismo nos encontramos con varios problemas que pasamos a desarrollar. Si bien es sabido de que el ciclo de vida de los procesos no es el estándar, que se indican en los libros más acreditados de la materia y que se encuentran al alcance de los alumnos, se pensó que la modificación del mismo podría resultar no tan compleja, sin embargo fue todo lo contrario porque al modificar totalmente la estructura ya el resto del sistema operativo no quedó estable, por lo que se decidió comenzar desde cero con un sistema operativo propio y de paso escribir nuestras experiencias como base para el estudio de la universidad.

1.2.- Área otros sistemas operativos de estudio.

Minix: Es un clon del sistema operativo Unix distribuido junto con su código fuente y desarrollado por el profesor Andrew S. Tanenbaum [TAN97] en 1987. La última versión oficial de Minix es la 3.1.2, publicada el 8 de Mayo de 2006.

Fue creado para enseñar a sus alumnos el diseño de sistemas operativos, poseía un reducido tamaño, basado en el paradigma del micronúcleo, y amplia documentación. Apropiado para personas que desean instalar un sistema operativo compatible con Unix en su máquina personal así como aprender sobre su funcionamiento interno. Minix fue desarrollado para correr sobre IBM PC con procesador Intel 8088 o superior y se ha portado a otros sistemas

Topsy: Es un pequeño sistema operativo el cual ha sido diseñado para propósitos de enseñanza (Topsy se refiere a Teachable Operating System), este entorno de trabajo permite la práctica de ejercicios relacionados con el curso de Computación II y un ejemplo de cómo los principios básicos pueden ser implementados en un sistema operativo real.

Nachos: Es un software instructivo que permite a los alumnos estudiar y modificar un sistema operativo real. La diferencia entre Nachos y un sistema operativo real es que Nachos corre como un simple proceso Unix y simula las facilidades de generales de bajo nivel de una máquina , incluyendo interrupciones, memoria virtual y manejo de dispositivos de entrada salida..

Zeus Os: El sistema está orientado a la sustitución de las prácticas que se realizan en la asignatura de Sistemas Operativos II con el fin de no solo aprender a nivel teórico, sino también de implementar un sistema totalmente funcional. El Objetivo fue trabajar sobre núcleos de otros sistemas operativos), como puede ser KMOS, para llevar a cabo su estudio y expansión.

Minirighi: Este sistema operativo multi-threading para arquitecturas IA-32, ofrece un kernel liviano, fácil de leer por todos y permite un análisis simplificado del código fuente a diferencia de la mayoría de lo otros sistemas operativos. Autor Andrea Righi.

Ninguno de los sistemas operativos, antes nombrados, permiten la parametrización de algoritmos en tiempo de ejecución, así como tampoco muestra los detalles de la ejecución de los algoritmos y sus estadísticas.

2. Introducción

El sistema operativo SODIUM es la consecuencia del proyecto de investigación sobre sistemas operativos de características didácticas que se lleva a cabo en la Universidad Nacional de la Matanza. El mismo está desarrollado por los alumnos que cursan la materia Sistemas Operativos transfiriéndose los conceptos desarrollados, año a año, a los alumnos que continuarán la tarea dejada por su pares de años anteriores. Hasta el momento se cuenta con el arranque y la instalación del mismo en cualquier máquina que tenga como base los estándares de IA-32, así como una interfaz de línea de comandos que permite interactuar con el sistema. Con el mismo, los alumnos pueden listar en pantalla todas las tablas internas del sistema, administrar procesos, realizar volcados de memoria, lanzar pruebas de concurrencia configuradas a gusto, y obtener las estadísticas de uso de CPU y tiempos de ejecución. Los resultados de las pruebas se almacenan en un archivo que oficia de historial, y podrá ser analizado posteriormente, presentando evidencias que son base para la comprensión definitiva del comportamiento de los algoritmos estudiados.

3. Lenguaje de programación

La mayor parte del proyecto se está desarrollando utilizando lenguaje ANSI C (más específicamente el compilador GCC), pero otra parte del código se desarrolla en lenguaje ensamblador, para llegar con más facilidad a todo aquel código de más bajo nivel.

Para llegar a esta opción el equipo de trabajo consideró los siguientes parámetros:

Es necesario un lenguaje al cual se lo considere lo suficientemente sólido y flexible.

Que permita una conexión simple con ASSEMBLER para manejar sentencias de bajo nivel y que el mismo compilador permita la interacción entre los dos lenguajes.

Un lenguaje que fuese conocido por los alumnos. ANSI C responde a este requerimiento ya que es el punto de partida de las materias de programación y que por lo tanto se aplica en otras materias.

Que fuese conocido por toda la comunidad universitaria. ANSI C responde a este requisito debido a que se enseña en las más prestigiosas carreras de informática.

4.- Importancia del código escrito

Es importante tener en cuenta que el SODIUM es un sistema operativo de estudio y por lo tanto no se mide en esta primera etapa el tiempo de ejecución del mismo, por lo tanto no se realizan funciones de optimización con la periodicidad que debieran realizarse y, como hemos consignado anteriormente, dado que este desarrollo está realizado por alumnos de la facultad para cumplimentar las exigencias de una materia y que además tienen su carga laboral, puede darse el caso de que existan más de una rutina o procedimiento repetido.

4.1.- Convenciones a la escritura

Para facilitar la tarea de unificación de los trabajos e implementaciones puntuales desarrolladas por los alumnos durante el transcurso del ciclo lectivo, fue necesario establecer una notación específica para la nomenclatura de funciones y variables, aplicables durante la programación en C y assembler. Esto a su vez permite al alumno familiarizarse en poco tiempo con la totalidad del sistema. Por citar un ejemplo, las variables no signadas comienzan con una letra “u” minúscula, y luego otra letra define el tipo de dato. Una variable entera sin signo comenzaría con “ui”. También se observan casos particulares que indican el propósito de la variable, como “p” para punteros, y “h” para identificadores de ventanas. El documento completo se encuentra disponible y totalmente desarrollado en [01].

5. Niveles de protección

Generalmente todos los sistemas operativos conocidos utilizan dos niveles de protección (los cuales coinciden con el Anillo 0 y 3 de la unidad de segmentación). Tanto Linux como Windows utilizan esos dos niveles, pero en SODIUM se utilizan todos los niveles del anillo INTEL porque la función que se persigue es el conocimiento, por lo tanto todos los alumnos deberán saber como se realiza el pasaje de anillo a anillo utilizando los GATES que correspondan en el momento que corresponda.

Sabemos positivamente que el proceso será más lento pero permitirá una mayor interiorización acerca de los mecanismos provistos por la arquitectura IA-32.

6. Estructura del Kernel

Después de un largo proceso de evaluación sobre cuál sería la estructura del sistema operativo para estudio, la estructura monolítica o la estructura por capas o niveles o jerárquica, se decidió por la estructura monolítica porque es la que más rápidamente permite una toma de conocimiento y de experiencia que faltaba en el equipo de trabajo y que la estructura por capas o niveles era más compleja de transmitir a aquellos que están incursionando por primera vez en la construcción de un sistema operativo.

Es la estructura de los primeros sistemas operativos constituidos fundamentalmente por un solo programa compuesto de un conjunto de rutinas entrelazadas de tal forma que cada una puede llamar a cualquier otra, es decir que tienen amplia libertad para comunicarse entre ellas.

La utilización de un procedimiento se llama directamente, no necesita mensajes, por lo que es más rápido y más fácil de conceptualizar.

Es sabido que todas las actualizaciones que se realicen en el futuro implican recompilar todo el núcleo, pero eso permite a los desarrolladores entender mejor la estructura del sistema manteniendo la configuración del Makefile.

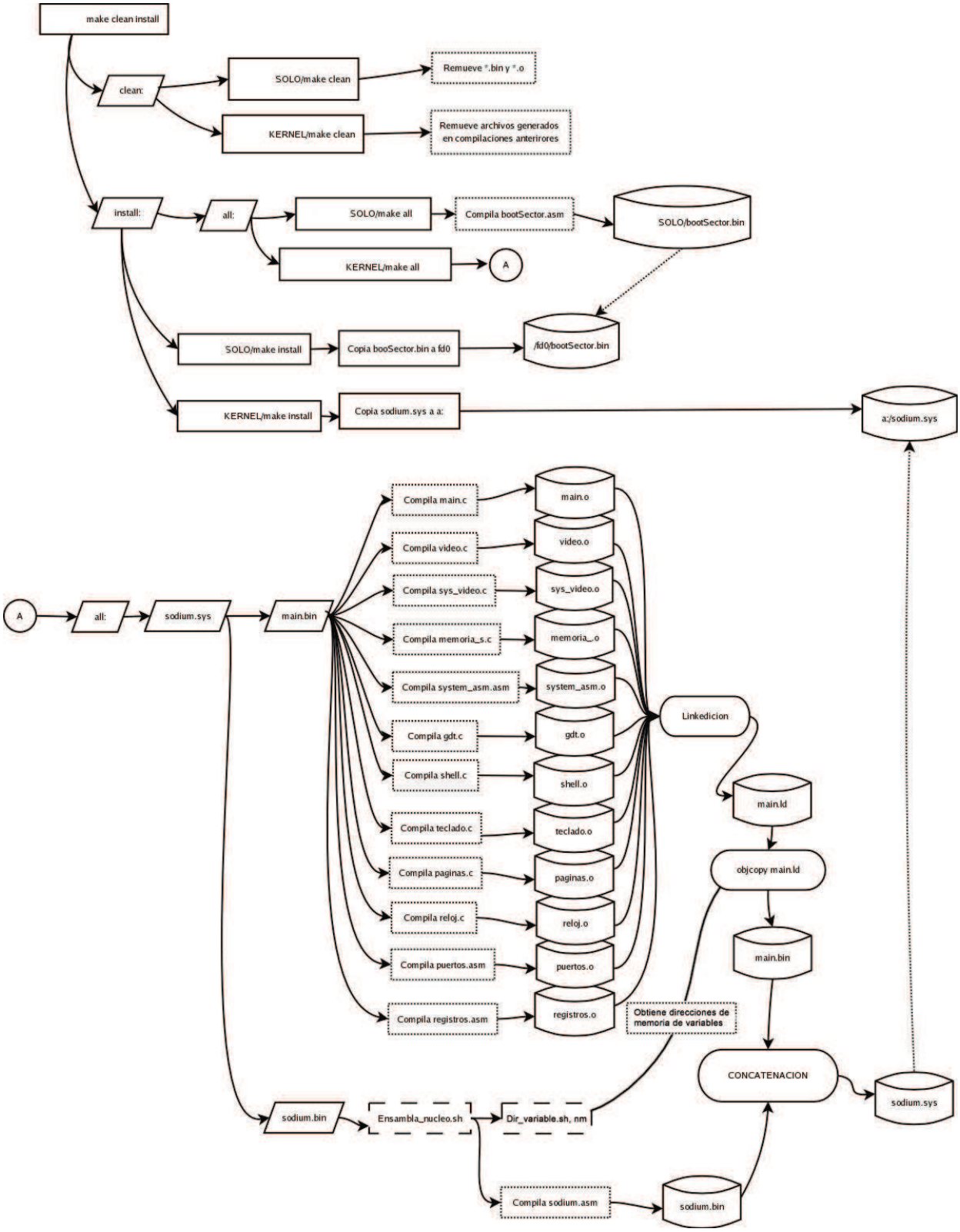
7. Plataforma de desarrollo

SODIUM se desarrolla exclusivamente con herramientas de código abierto. A continuación se detalla las versiones de los programas y utilitarios más antiguos con los que se sabe que se puede compilar el SODIUM:

Kernel	2.2.5-15	Linux RedHat 6.0, por ejemplo
Nasm	0.98.38	NetWide Assembler, compilador con soporte para notación Intel
GCC	2.91.66	Compilador C de GNU que genera archivos objeto.
DD	4.0	(GNU fileutils) Comando que copia un archivo (o stdin/out)
LD	2.9.1	Linkeditor
CAT	1.22	(GNU textutils) Concatena archivos
MAKE	3.77	Construye proyectos en base a árboles de dependencia de compilación
OBJCOPY	2.9.1	Utilidad que copia el contenido de un archivo objeto a otro tipo

También ha sido compilado correctamente en otras distribuciones más actuales, como Ubuntu, Kubuntu, Fedora, Slackware y Knoppix.

8. Diagrama de compilación, enlazado, y armado del disquete de pruebas.



9. Proceso de Carga

El SODIUM inicia ejecutando en modo real, en un entorno pre-establecido por la BIOS.

Durante la primera etapa de carga es necesario establecer el contenido de los registros de la CPU a valores conocidos, y evitar el uso de áreas de memoria antes de inicializarlas. Como es sabido, no todas las BIOS se comportan de la misma manera, de modo que durante el proceso de carga es necesario, por ejemplo, establecer mediante un salto largo los valores correctos de segmento y offset a 0x0000:(0x7C00.+ dirección_salto) para que nuestro código se ejecute correctamente.

El SODIUM ocupa relativamente poco espacio (<500kbytes), de manera que es posible ejecutarlo directamente desde un disquete, aunque también puede ser instalado y ejecutado desde un disco rígido, contando con un multi-bootloader para compatibilidad con otros sistemas previamente instalados, y los sectores de booteo correspondientes a cada medio físico. Por lo tanto, el proceso de carga varía levemente de acuerdo al marco de trabajo elegido.

Por razones de compatibilidad y simplicidad, hemos elegido, en esta primera etapa, soportar los sistemas de archivos FAT12 y FAT16. Utilizamos FAT12 en el disquete, y FAT16 en la partición del disco rígido.

Si el primer dispositivo accesible de booteo es un disco rígido, la BIOS copiará su primer sector (Master Boot Record) en la dirección lineal 0x7C00, y si el mismo contiene una firma válida (0xAA55), le cederá la ejecución. Si se trata de nuestro MBR, entonces mostrará en pantalla el listado de particiones válidas que posee el disco rígido, y nos permitirá seleccionar una de ellas mediante el teclado (eligiendo un número del 1 al 4). Nuestro MBR se reubica en la dirección lineal 0x600h y carga en la dirección 0x7C00 al primer sector correspondiente a la partición seleccionada, cediéndole luego la ejecución únicamente si posee la firma correcta (0xAA55). En nuestro caso se cede la ejecución al sector de booteo que trabaja con FAT16.

Si el primer dispositivo accesible de booteo es un disquete, la BIOS copiará su primer sector (sector 0 o sector de booteo) en la dirección lineal 0x7C00, y si el mismo contiene una firma válida, le cederá la ejecución. En nuestro caso se cede la ejecución al sector de booteo que trabaja con FAT12.

A partir de ese momento comenzamos el proceso de carga de nuestro SO, que se realizará en dos etapas.

9.1.- Primera etapa de carga (Sector de Booteo):

Consiste en establecer un stack válido, y copiar el archivo loader.sys desde el sistema de archivos del medio físico hacia la memoria. Para ello es necesario leer el directorio raíz y obtener el número de cluster inicial de dicho archivo. Luego se lee el archivo, en memoria, siguiendo cluster a cluster la cadena en la FAT (File Allocation Table) que corresponde al mismo. Cabe destacar que en un disquete de 1.44mb aloca siempre 1 sector por cluster, mientras que un disco rígido puede alocar de 2^n sectores por cluster, con n de 0 a 7. Una vez que se finaliza la carga de dicho archivo a partir de la dirección lineal 0x7e00, se le cede la ejecución, realizando un salto largo al inicio del mismo, y comienza la segunda etapa de carga.

9.2.- Segunda etapa de carga (Loader.sys)

La función del loader es la de recorrer un listado de archivos predefinido en tiempo de compilación, y copiarlos en las posiciones de memoria deseadas. Las rutinas de acceso al sistema de archivo son similares a las utilizadas durante la primera etapa, pero aquí se les da un uso más extenso y versátil.

El objetivo de esta implementación es, por un lado, continuar el proceso de carga, y por el otro, permitir a los alumnos la compilación y posterior ejecución de sus propios programas de prueba.

El listado puede verse de la siguiente manera, asumiendo dos archivos de prueba:

```
ListaDeArchivos:
db "SODIUM  SYS"    ;Nombre
db 0x12345678;Firma Inicial
db 0x87654321;Firma Final
dw 0x07E0           ;Segmento
dw 0x0000           ;Offset

db "MBR     SYS"    ;Nombre
db 0x2468ACE0;Firma Inicial
db 0x13579BDF;Firma Final
dw 0x5000           ;Segmento
dw 0x0000           ;Offset

db "PRUEBA1 SYS"    ;Nombre
db 0x11111111;Firma Inicial
db 0x22222222;Firma Final
dw 0x6000           ;Segmento
dw 0x0000           ;Offset

db "PRUEBA2 SYS"    ;Nombre
db 0x33333333;Firma Inicial
db 0x44444444;Firma Final
dw 0x7000           ;Segmento
dw 0x1000           ;Offset
FinListaDeArchivos:
```

Antes de comenzar efectivamente la carga en memoria de los archivos presentes en el listado, el loader se reubica a sí mismo a partir de la dirección lineal 0x90000. El primer archivo (sodium.sys) es parte integral del sistema operativo y siempre debe ser cargado.

El último paso del proceso de carga es el de ceder la ejecución al inicio del archivo sodium.sys.

10. Proceso de Inicialización del SODIUM

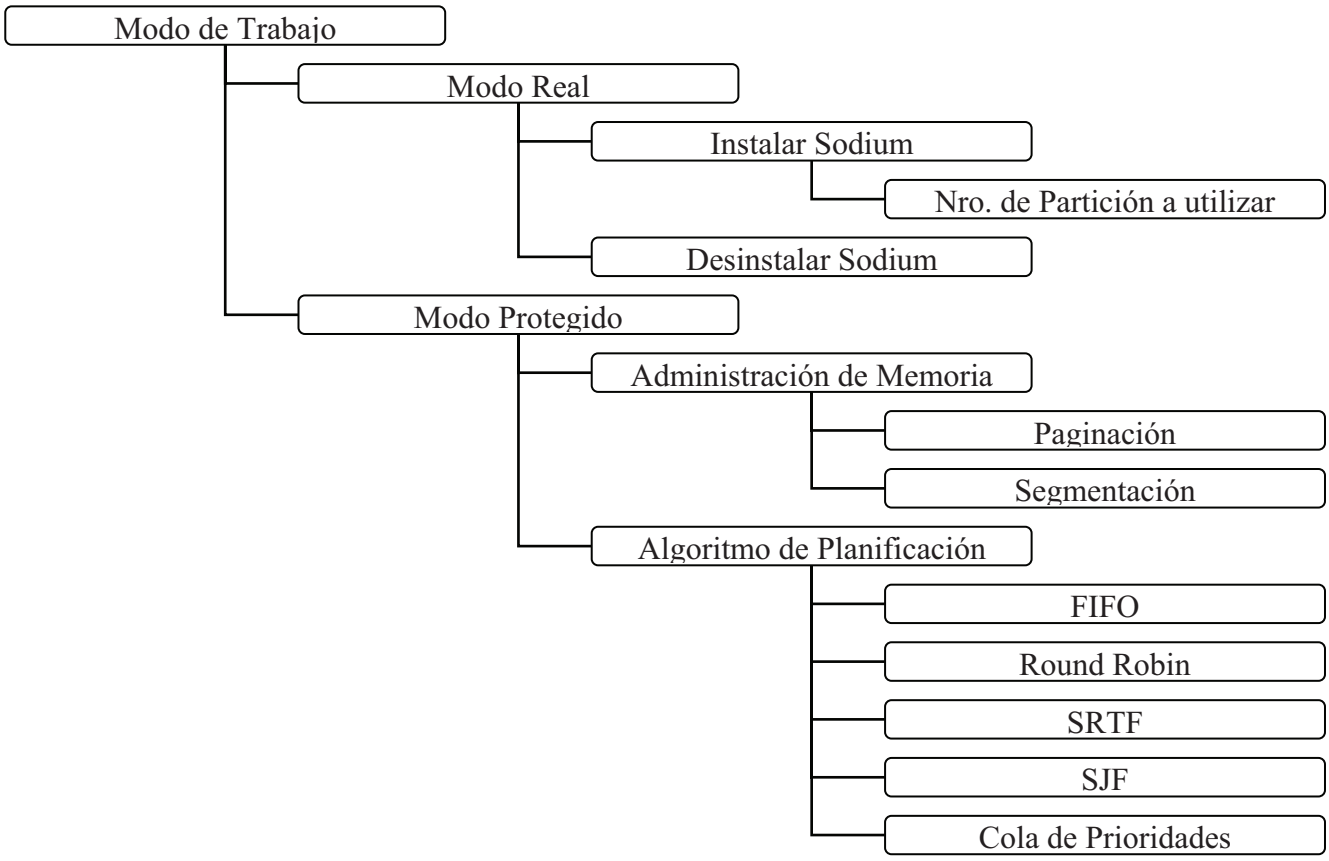
El archivo sodium.sys está compuesto por dos partes compiladas independientemente y luego concatenadas. La primer mitad es código binario ejecutable de 16bits, programado en assembler, y la segunda mitad consiste en el kernel propiamente dicho. Este último se compone exclusivamente de código de 32bits, está programado en lenguaje C y se ejecuta siempre en modo protegido. El orden de compilación de estos bloques está pensado para que durante la ejecución del código en modo real se conozca la posición de distintas variables en el kernel, y de esta forma se establezca una comunicación sencilla de las opciones de inicialización elegidas.

El proceso de inicialización también se realiza en dos etapas, culminando con la activación del planificador de tareas, que es el momento a partir del que la interfaz de línea de comandos puede leer datos del buffer.

10.1.- Primera etapa de Inicialización

Inicialmente se obtiene la cantidad de memoria base y extendida de la PC por medio de los servicios de la BIOS, y se copian dichos valores en las variables globales respectivas del kernel.

Utilizando los servicios de la bios para el manejo de pantalla y teclado, se presenta una serie de ventanas de configuración, que solicitan al usuario la elección inicial del modo de operación del SODIUM. Aquí se permite navegar por las siguientes opciones:



Una vez que el usuario está satisfecho con la configuración inicial, se copian los valores establecidos a las variables globales del kernel.

Luego se procede a habilitar la compuerta A20. Esto es necesario para poder realizar, una vez que pasemos a modo protegido, direccionamientos de más de 1MB. Dado que sabemos positivamente que no todas las computadoras aceptan el mismo mecanismo para habilitar dicha compuerta, utilizamos cuatro de los métodos más difundidos hoy en día, uno por vez, hasta verificar que el pasaje se efectuó correctamente.

Posteriormente se reubica el código de inicialización hacia una dirección más alta en memoria, cerca de los 640KB, de manera de disponer de todo el espacio que sea posible para reubicar el kernel a partir de la dirección lineal 0x0. Nótese que en este proceso estamos sobreescribiendo la IVT (Interrupt Vector Table) y la BDA (BIOS Data Area), de manera que es importante que esta copia se ejecute con las interrupciones inhibidas. De hecho, no se volverán a utilizar hasta que se termine la segunda etapa de inicialización.

Luego se inicializa a 0x0 un segmento de 64KB de memoria, contiguo a la BSS (Block Started by Symbol), que se utilizará para contener la GDT (Global Descriptor Table), y se cargan los primeros tres descriptores de la misma con el descriptor nulo, un descriptor de código, y uno de datos. Estos dos últimos, están configurados de manera que permitan direccionar 4GB a partir del inicio de la memoria, con un DPL (Descriptor Privilege Level) de 0.

Por último se establece un nuevo stack al final de la memoria convencional, y se realiza el pasaje a modo protegido seteando el bit PE del registro CR0 de la CPU y ejecutando un salto largo a la dirección lineal 0x0, usando CS (Code Selector) = 8.

10.2.- Segunda etapa de Inicialización

Aquí ya nos encontramos ejecutando en modo protegido. El proceso de inicialización ahora corresponde a la secuencia preparada en el main.c del kernel. Cada operación se confirma en la pantalla con un mensaje de "[HECHO]" u "[ERROR]", y se muestran en pantalla las elecciones realizadas durante la etapa anterior de inicialización.

Dado que al trabajar en modo protegido necesitaremos manejar la totalidad de las excepciones de la CPU, es necesario reprogramar el PIC (Programmable Interrupt Controller) para reubicar las interrupciones de hardware a partir del índice 0x20.

Se programa el Timer, indicándole que comience a provocar interrupciones cada 0xFF ticks. Elegimos el máximo intervalo posible de tiempo entre interrupciones para que el mismo sea apreciable a la vista, una vez que el sistema está en funcionamiento.

Se inicializa la IDT (Interrupt Descriptor Table) con manejadores para las excepciones 0x0 a 0x1A, para las interrupciones de timer (0x0), teclado (0x1), disquetera y disco rígido, y por último se instala el vector de atención de pedidos de servicio (SysCalls) en la posición 0x80. Se le instruye al PIC para que desenmascare las interrupciones de estos dispositivos físicos.

De acuerdo al modo de administración de memoria se inicializa una tabla de segmentos por proceso o un mapa de bits y una tabla de directorios de página.

Luego se verifica que los archivos copiados a memoria durante el proceso de carga se encuentren en las direcciones de memoria deseadas. Para esto se busca que existan en memoria 2 firmas prefijadas de 32 bits de longitud por cada archivo, las cuales son colocadas al inicio y al fin de los mismos durante el proceso de compilación.

Se inicializa un mapa de bits que mantiene un seguimiento de las entradas libres de la GDT, para asignarlas a medida que se crean tareas, y se inicializa el vector estático de PCBs (Process Control Blocks). Posteriormente se crean tres tareas, la tarea nula, la tarea shell, y la tarea reloj. Dado que estas tres son funciones del kernel, sus respectivos selectores de código, datos y stack apuntan a los descriptores del kernel. Para cada una se crea un descriptor de tareas (Task State Segment) independiente.

- La tarea "Nula" ocupará el procesador mientras no hay otros procesos compitiendo por él.
- La tarea "Reloj" tomará la fecha y hora del RTC (Real Time Clock) y la imprimirá en pantalla.

- La tarea “Shell” se encargará de refrescar la línea de comandos con los caracteres ingresados por el teclado, manejar el historial, e interpretar los mismos una vez que se presiona la tecla “Enter”.

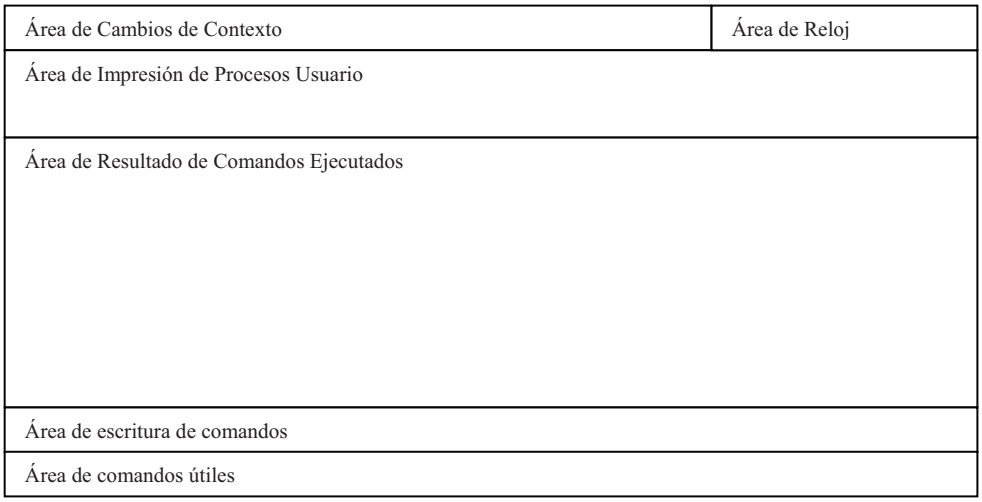
Por último se carga el registro TR (Task Register) con una posición de la TSS (Task State Segment) que puede utilizar para descargar el contenido de los registros del procesador al momento de ejecutar el primer cambio de contexto.

Una vez que se activa el planificador y se habilita la atención de interrupciones, el sistema se torna interactivo.

11. Presentación de la Interfaz

La interfaz de SODIUM es sencilla. Trabaja en modo terminal, con una matriz de 80x25 caracteres, y soporta una paleta de 16 colores. La escritura en pantalla se logra accediendo directamente a la memoria de video, que se encuentra mapeada a partir de la dirección 0xB8000. La pantalla se encuentra dividida en distintas áreas con propósitos específicos, como se ilustra a continuación.

11.1.- Distribución de las áreas en pantalla



Área de cambio de contexto: Es una línea donde el planificador escribe la información adecuada para identificar al proceso que está ejecutando actualmente. Esta es: PID (Process Id), Nombre del Programa, y el índice correspondiente en la GDT.

Área Reloj: Es una línea donde la tarea “Reloj” escribe la hora del sistema.

Área de Impresión de Procesos Usuario: Es un área de 5 líneas donde se permite que los procesos usuario impriman mensajes útiles, como el valor de contadores internos, cantidad de ciclos realizados, cantidad de operaciones E/S, etcétera. Esta ventana puede ser ocultada a voluntad, para maximizar la visualización en pantalla de los resultados de comandos ejecutados por el usuario.

Área de Resultado de Comandos Ejecutados: De acuerdo al comando ejecutado, se presenta el resultado en pantalla, que puede constar de una o cientos de líneas, como dumps de memoria, visualización de tablas del sistema, como la GDT, IDT, bitmaps, etcétera.

Área de Escritura de Comandos: Esta área puede extenderse si el comando ingresado ocupa más de una línea de longitud. Soporta dos métodos de autocompletado de comandos, por tabulación o automático por mayor similitud. Los comandos pueden recibir una cantidad variable de parámetros y los números pueden ser escritos tanto en notación decimal como hexadecimal.

También se dispone de un historial circular de hasta 20 comandos, de manera que utilizando las flechas cursoras hacia arriba o hacia abajo se puede ejecutar instrucciones tipeadas con anterioridad.

Área de Comandos Útiles: En esta única línea se listan los comandos más utilizados durante las pruebas.

12. Comandos del SODIUM

12.1.- Descripción de los comandos generales

Comando	Descripción
Ayuda	Muestra una descripción en pantalla de los comandos del SODIUM y sus parámetros.
bitmap	Muestra el mapa de memoria libre y alocada.
check [pid] [pag] [offset]	Verifica si para el proceso dado los valores de página y offset caen dentro de los límites de memoria que le fueron asignados. En caso contrario lo finaliza.
cls	Limpia la ventana de ejecución de comandos.
desc [indice]	Muestra los campos en detalle que componen al descriptor por el cual se consulta.
dump [desde] [longitud]	Realiza un vuelco de memoria en pantalla. Indicar Dirección inicial y cantidad de words a mostrar. La salida se muestra paginada.
exec [opcion] [tamaño] [prioridad]	Se crea un proceso que comparte el espacio de direcciones del kernel. Las opciones posibles son 1 y 2, siendo 1 un proceso que utiliza el 100% de CPU y 2 un proceso que imprime un string en pantalla mediante un syscall. Se indica el tamaño reservado de memoria en bytes. Esta información se utiliza para la simulación de asignación de memoria paginada y segmentada. Para los algoritmos que la utilizan, se indica una prioridad de ejecución.
execve [nombre_archivo] [parámetros] [tam_bss] [prioridad]	Se crea un proceso a partir de un archivo de prueba ejecutable compilado de forma independiente. Es necesario indicar el tamaño de BSS que resultó de dicha compilación. Para este proceso se crea un espacio de direccionamiento acotado, de modo que no puede acceder directamente a memoria de video ni a las funciones del kernel, salvo por los mecanismos de SysCalls. Los parámetros adicionales hacia el proceso se envían de la forma tradicional a través del stack. Para los algoritmos que la utilizan, se indica una prioridad de ejecución.
gdt	Muestra el contenido de las primeras 20 posiciones en la GDT.
idt	Muestra el contenido de la IDT.
kill [pid]	Remueve un proceso de la memoria. Indicar pid.
ls [dispositivo]	Muestra los archivos presentes en el directorio actual.
mem	Muestra la cantidad de Memoria base y total y disponible del sistema, y estadísticas de uso y fragmentación de la misma.
pag [pid]	Muestra los frames correspondientes a ese proceso.
ps	Lista los Procesos cargados en memoria e indica el estado de cada uno.
reboot	Efectúa un reinicio en caliente de la la PC.
segs	Muestra el nro, posición inicial y posición final de los segmentos de memoria ocupados.
set [variable] = [valor]	Se utiliza para establecer o modificar el valor de variables de entorno globales.
stack [pid] [ring 0-3]	Muestra el contenido del stack para el proceso y ring de ejecución especificado.
syskill [pid] [sig]	Envía una señal a un proceso determinado. Actualmente las señales soportadas son SIGSTOP, SIGCONT, SIGKILL, SIGTERM, SIGALRM
tss [pid]	Muestra el contenido de la TSS y PCB del proceso indicado.
ver	Indica la versión del sistema operativo.
winm	Oculta o restaura la ventana de procesos de usuario.

12.2.- Mecanismos de planificación disponibles

Comando	Descripción
planif [subcomando] [parámetros]	Bajo este comando se agrupan otros subcomandos que permiten establecer y configurar el planificador de turno.
planif aplicar	Aplica los cambios a las variables del planificador, reiniciando al mismo.
planif cambiar [nombre_algoritmo]	Establece el nuevo algoritmo de planificación a utilizar.
planif defecto	Establece los parámetros por defecto para el planificador actual.
planif mostrar	Muestra en pantalla el algoritmo utilizado actualmente y el valor de los parámetros configurables del mismo.
planif set [parametro] = [valor]	Cambia el valor de un parámetro utilizado por el algoritmo de planificación. Dichos cambios no tendrán efecto hasta que se utilice el subcomando "aplicar"

12.3.- Ejecución de pruebas por lotes

Comando	Descripción
lote [subcomando] [parámetros]	Bajo este comando se agrupan otros subcomandos que permiten administrar los Lotes de Prueba de ejecución.
lote ayuda	Se muestra la lista de comandos y su descripción.
lote cargar [nombre_archivo]	Lee un archivo descriptor de lote a memoria, y devuelve el nuevo ID
lote editar [ID]	Permite editar un lote de pruebas, seteando nuevos archivos y condiciones de ejecución, en lenguaje LCL
lote eliminar [ID]	Elimina un lote de pruebas existente.
lote guardar [ID] [nombre_archivo]	Guarda un lote de pruebas en un archivo de texto determinado.
lote iniciar [ID]	Inicia la ejecución de un lote de pruebas.
lote ls	Lista los lotes que han sido creados durante la sesión actual.
lote mostrar [ID]	Muestra las operaciones almacenadas en el lote de pruebas.
lote nuevo	Crea un nuevo lote de pruebas y muestra en pantalla su ID
lote pausar	Se detiene momentáneamente todos los procesos lanzados.
lote reanudar	Se reanuda a todos los procesos del lote de pruebas actual, si este fue pausado anteriormente.

12.4.- Extracción de los resultados de las pruebas para su posterior evaluación

Comando	Descripción
log [subcomando] [parámetros]	Bajo este comando se agrupan otros subcomandos que permiten administrar el registro.
log ayuda	Se muestra la lista de comandos y su descripción.
log eliminar	Inicializa a 0 la memoria reservada para la toma de eventos
log finalizar	Finaliza la toma de eventos.
log guardar [nombre_archivo]	Permite exportar el log a un archivo de texto con un nombre determinado en un disquete o en el disco rígido.
log iniciar	Comienza a guardar eventos en el log
log ls	Muestra el log actual en pantalla
log pausar	Se detiene momentáneamente la toma de eventos.
log reanudar	Se reanuda la toma de eventos, si esta fue pausada anteriormente.
log tamanio [Knro_eventos]	Establece el tamaño del log en múltiplos de 1024 eventos.

13. Conclusiones

De acuerdo a lo expuesto el sistema operativo SODIUM se convierte en una herramienta útil para los alumnos y profesores de las materias relacionadas. De él pueden obtenerse excelentes comparaciones para el análisis de los algoritmos más importantes. Hasta el momento sólo se relaciona con la administración de procesos, pero todo el equipo de investigación se encuentra abocado a generar la posibilidad de comparar algoritmos de administración de memoria y de administración de entrada salida generando los consecuentes drivers, manteniendo la característica de la parametrización

14. Bibliografía

[ANG98] Angulo José M. y Funke Enrique – Microprocesadores avanzados 386 y 486 – Introducción al Pentium y Pentium – Pro Editorial Paraninfo – Cuarta Edición

[BRE00] BRE00- Brey Barry B. – Los Microprocesadores Intel – Editorial Prentice Hall – Quinta Edición.

[CAR01] Card Rémy, Dumas Eric, Mével Franck - Programación Linux 2.0 API del sistema y funcionamiento del núcleo – Enrolles y Ediciones Gestión 2000 S.A.

[DEI90] Deitel Harvey M. – Introducción a los Sistemas Operativos - Addison-Wesley Iberoamericana – Segunda Edición

[INTEL] Manual de microprocesadores 386 y 486 y Pentium.

[MIL94] Milenkovic Milan – Sistemas Operativos Conceptos y diseño – Mc Graw Hill – Segunda edición

[SIL97] Silverschatz, Avi; Galvin, Peter – Operating System Concepts – Addison-Wesley Longman – Fifth Edition

[SMC00] Standard Microsystems Corporation – Application note 6.12

[STA98] Stallings Willams – Operating Systems Internals and design principles – Prentice Hall International – Third Edition

[TAN97] Tanenbaum Andrew S., Woodhull Albert S. – Operating Systems Design and Implementation – Prentice Hall – Second Edition

[TAN03] Tanenbaum Andrew S.– Sistemas Operativos Modernos – Pearson Education – SegundaEdición

[TUR03] Turley James L. – Advanced 80386 programming techniques – Osborne McGraw Hill

Internet

[01] Página de la cátedra www.souniver.com.ar

[02] INTEL 8272 Floppy disk Controller
http://andercheran.aiind.upv.es/~amstrad/docs/i8272/8272sp.htm

[03] BONAFIDE os development (detecting floppy drives)
http://www.osdever.net/tutorials/detecting_floppy_drives.php?the_id=58

[04] BONAFIDE os development (how to program the DMA)
http://www.osdever.net/tutorials/howto_dma.php?the_id=63

[05] The Unix File System
http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html

[06]Cátedra de la profesora Gloria Guadalupe González Flores de la Universidad Juárez Autónoma de Tabasco trabajo sobre DMA de Roberto García García
http://mx.geocities.com/antrahxg/documentos/org_comp/procesador.html#inicio

[07] Tecnología del PC- La Placa base http://www.zator.com/Hardware/H2.htm