Proyecto Sodero

Introducción

Presentación

El **Proyecto Sodero** es una idea que surgió de un grupo de amigos, todos alumnos de la Universidad Nacional de La Matanza con el objetivo de realizar lo que mas nos gusta: **Programar** y poder adquirir nuevos conocimientos como son programar en lenguaje ensamblador y saber algo más sobre teoría de Sistemas Operativos.

Nuestro objetivo principal, ademas del de aprender, se centra en el desarrollo de un **Sistema Operativo**, abarcando toda la funcionalidad que el mismo requiere.

¿Por qué Proyecto Sodero?

Como comentábamos antes, todos somos alumnos de la Universidad Nacional de La Matanza, y a la vez, estamos cursando, dentro de la carrera, la orientación **Robótica**, con lo cual el nombre de nuestro "futuro" Sistema Operativo es un acrónimo de:

SODERO = **S**istema **O**perativo **de RO**bótica

Comienzos

Si bien la inquietud sobre poder crear **nuestro propio Sistema Operativo** es algo que existió desde que empezamos la carrera, fue a partir de cursar la materia "Sistemas de Computación II" que dicha inquietud comenzó a hacerse más fuerte y no tan utópica como nos parecia hasta llegar a mes de febrero de 2002, en donde finalmente pudimos concretar nuestro viejo anhelo.

Es entonces, el mes de febrero de 2002 la fecha de nacimiento de este proyecto que esperemos con el correr de los meses pueda concretarse de la mejor forma, como los cuatro integrantes del grupo tanto deseamos.

Introducción

Presentación

El **Proyecto Sodero** es una idea que surgió de un grupo de amigos, todos alumnos de la Universidad Nacional de La Matanza con el objetivo de realizar lo que mas nos gusta: **Programar** y poder adquirir nuevos conocimientos como son programar en lenguaje ensamblador y saber algo más sobre teoría de Sistemas Operativos.

Nuestro objetivo principal, ademas del de aprender, se centra en el desarrollo de un **Sistema Operativo**, abarcando toda la funcionalidad que el mismo requiere.

¿Por qué Proyecto Sodero?

Como comentábamos antes, todos somos alumnos de la Universidad Nacional de La Matanza, y a la vez, estamos cursando, dentro de la carrera, la orientación **Robótica**, con lo cual el nombre de nuestro "futuro" Sistema Operativo es un acrónimo de:

SODERO = **S**istema **O**perativo **de RO**bótica

Comienzos

Si bien la inquietud sobre poder crear **nuestro propio Sistema Operativo** es algo que existió desde que empezamos la carrera, fue a partir de cursar la materia "Sistemas de Computación II" que dicha inquietud comenzó a hacerse más fuerte y no tan utópica como nos parecia hasta llegar a mes de febrero de 2002, en donde finalmente pudimos concretar nuestro viejo anhelo.

Es entonces, el mes de febrero de 2002 la fecha de nacimiento de este proyecto que esperemos con el correr de los meses pueda concretarse de la mejor forma, como los cuatro integrantes del grupo tanto deseamos.

Objetivos del Proyecto

Comentarios previos

En el momento de comenzar con el proyecto nos propusimos **muchos** objetivos, en los párrafos subsiguientes vamos a intentar enumerarlos a todos, aunque es importante aclarar que no tenemos un orden de prioridades para los mismos, si bien del éxito del proyecto depende en forma directa nuestro título, consideramos que tan importante como eso es llegar al final con todos los conocimientos necesarios asimilados de la mejor forma posible, aunque eso signifique no finalizar este año con nuestro proyecto.

Objetivos y Metas

- Poder desarrollar **integramente**, partiendo desde cero, nuestro propio Sistema Operativo. Compuesto por un kernel monolítico, un conjunto de llamadas al sistema y un shell que permita operarlo. El mismo correrá en modo protegido, será multitarea y monousuario.
  Vea la sección **documentación** para un desarrollo teórico sobre estos conceptos y sobre las características de **SODERO**.
- Comprender y dominar todos los conceptos relacionados con la Teorí de Sistemas Operativos, indispensable para el éxito del proyecto.
- Comprender y dominar la arquitectura Intel para poder desarrollar aplicaciones eficientes en modo protegido bajo dicha plataforma.
- Debido a la escasa informacion técnica encontrada sobre el tema de nuestro interés, consideramos muy importante la creación en forma paralela al desarrollo del Sistema Operativo, de la documentación necesaria para que cualquiera que lo desee pueda contar con toda la información que en su momento nosotros tanto hubiésemos deseado.
- Brindarle nuestro "granito de arena" a la comunidad GNU desarrollando bajo dicha licencia la totalidad de nuestro proyecto para que cualquier persona sea **libre** de ver, mejorar y contribuir al crecimiento del **Proyecto Sodero**.
- Finalmente, no dejamos de lado la meta de **aprobar** la materia Proyecto, condición indispensable para que podamos obtener nuestro título.

Un agradecimiento especial al profesor **Jorge Doorn**, jefe de cátedra de la materia "Lenguajes y compiladores", por brindarnos su ayuda desinteresada en la primer etapa del proyecto.

Modo Protegido

Introducción

El direccionamiento de memoria en modo protegido (a partir del 80286 y posteriores) permite acceder a datos y programas localizados por encima y dentro del primer megabyte de memoria. Para direccionar esta sección extendida el sistema de memoria se requiere un cambio en el esquema de direccionamiento de segmento más desplazamiento usado en el modo real. Cuando los datos y programa se direccionan la memoria extendida, se sigue utilizando la dirección de desplazamiento para acceder a la información en el segmento de memoria. Una diferencia consiste en la dirección del segmento ya que no existe en modo protegido. En lugar de una dirección de segmento, el registro de segmento contiene un SELECTOR que elige un descriptor de una tabla.

El descriptor especifica la ubicación del segmento en memoria, su longitud y sus derechos de acceso. Dado que el registro de segmento y la dirección de desplazamiento aún acceden a la memoria, las instrucciones del modo protegido son idénticas a las de modo real. De hecho, la mayoría de los programas escritos para funcionar en modo real funcionarán sin cambios en el modo protegido.

La diferencia entre los dos modos es la forma en que el microprocesador interpreta el registro de segmento para acceder al segmento de memoria. Otra diferencia, en los 80386 y posteriores, es que en modo protegido la dirección de desplazamiento puede ser un número de 32 bits en vez de utilizar uno de 16 bits como en modo real. Es por esto que puede direccionar hasta 4 Gb de longitud.

El SELECTOR, ubicado en el registro del segmento, elige uno de 8192 descriptores en una de las dos tablas de descriptores. El DESCRIPTOR especifica la ubicación, la longitud y los derechos de acceso del segmento de memoria, aunque no directamente como en el modo real. Por ejemplo, en el modo real, si CS=0008H, el segmento de código inicia en la localidad 00080H. En modo protegido, este número de segmento puede direccionar cualquier localidad de memoria en todo el sistema para el segmento de código.

Existen dos tablas de descriptores utilizadas con los registros de segmentos: una contiene descriptores globales y otra, descriptores locales. Los descriptores globales contienen las definiciones de los segmentos que se aplican a todos los programas, mientras que los descriptores locales son generalmente exclusivos de una aplicación. Podríamos llamar descriptor de sistema a un descriptor global, y descriptor de aplicación a uno local. Cada tabla de descriptores contendrá 8192 entradas, por lo tanto una aplicación podría disponer en cualquier momento de 16.384 descriptores. Puesto que un descriptor describe un segmento de memoria, esto permite que puedan ser descriptos hasta

16.384 segmentos de una aplicación.

Descriptor del 80286

7	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	6
5	Derechos de Acceso	Base (B23-B16)	4
3	Base (B15-B0)		2
1	Limite (L15-L0)		0

Descriptor del 80386 al Pentium II

7	BASE (B13-B24)	G	D	O	A V	LIMITE (L19-L16)	6
5	Derechos de Acceso	Base (B23-B16)					4
3	Base (B15-B0)						2
1	Limite (L15-L0)						0

Llamaremos DIRECCION DE BASE a la parte del descriptor que indica el inicio de un segmento de memoria. Para el microprocesador 80286, la dirección base es de 24 bits, así que los segmentos inician en cualquier localidad dentro de sus 16 MB de memoria. Los 80386 y posteriores emplean una dirección de base de 32 bits, que permite que los segmentos incien en cualquier localidad de sus 4 Gb de memoria. Es por lo dicho anteriormente que podemos ver que el descriptor del 80286 es compatible con el 80386 al Pentium II, ya que sus 16 bits más significativos son los 0000H.

El LIMITE DE SEGMENTO contiene la última dirección de desplazamiento del segmento.

A partir del 386 hasta el Pentium se incluyó el bit G o llamado BIT DE GRANULARIDAD. Si este G=0, el límite especifica un límite de segmento entre 00000H y FFFFF. Si G=1, el valor del límite se multiplica por 4 KB (agregando XXXXH). El límite para G=1 va entonces desde 00000XXXH hasta FFFFFXXXH. Esto permite una longitud de segmento de 4 KB a 4 GB en intervalos de 4 KB.

En el descriptor del 80386 y posteriores, el bit AV es usado por algunos sistemas operativos para indicar que el segmento esta disponible (AV=1) o que no lo esta (AV=0). El bit D indica la forma en que las intrucciones de los microprocesadores 80386 al Pentium II acceden a los registros y a los datos de memoria tanto en modo protegido como en el real. Si D=0, las instrucciones son de 16 bits,

compatibles con los microprocesadores 8086 al 80286. Esto significa que las instrucciones utilizan registros y direcciones de desplazamiento de 16 bits en forma predeterminada. Si D=1, las instrucciones son de 32 bits. Por omisión, el modo de instrucciones de 32 bits da por hecho que todas las direcciones de desplazamiento y los registros son de 32 bits.

7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

A=0 El segmento no ha sido accedido
A=1 El segmento ha sido accedido

E=0 El descriptor describe a un segmento de datos

ED=0 El segmento se expande hacia arriba (segmento de datos)
ED=1 El segmento se expande hacia abajo (segmento de pila)

W=0 Los datos no pueden se escritos
W=1 Los datos pueden se escritos

E=1 El descriptor describe a un segmento de código

C=0 Ignora el nivel de privilegio del descriptor
C=1 Honra el nivel de privilegio

R=0 El segmento de código no puede leerse
R=1 El segmento de código puede leerse

S=0 Descriptor de sistema
S=1 Descriptor de segmento de código o datos

DLP = Establece el nivel de privilegio del descriptor

P=0 Descriptor sin definir
P=1 El segmento contiene una base y límite válidos.

En modo protegido, el byte de derechos de acceso controla el acceso al segmento de memoria. Este byte define el funcionamiento del segmento en el sistema. El byte de derechos de acceso permite un control completo sobre el segmento. Si el segmento es de datos, especifica el sentido de crecimiento; si el segmento crece más allá de su límite, el programa del microprocesador es interrumpido, indicando un error de protección general. Se puede especificar si un segmento de datos puede ser escrito o si está protegido contra escritura. E segmento de código es controlado en forma similar y su lectura puede ser inhibida para proteger el software.

Los descriptores son seleccionados mediante el registro de segmento de la tabla de descriptores.

El registro de segmento contiene un campo de selección de 13 bits, un bit selector de tabla y un campo de nivel de privilegio de acceso, El selector de 13 bits selecciona uno de los 8192 descriptores de la tabla. El bit TI elige entre la tabla de descriptores globales (TI=0) o los descriptores locales (TI=1). El RPL (nivel de privilegio de acceso) define el nivel de privilegio de un segmento de memoria. El nivel de privilegio más alto es 00 y el más bajo 11. Si el nivel de privilegio solicitado coincide o tiene mayor prioridad que el nivel de privilegio establecido por el byte de derechos de acceso, se permite el acceso. Por ejemplo si el nivel de privilegio asociado es 10 y el byte de derechos de acceso establece un nivel de privilegio de 11 para el segmento, el acceso es autorizado ya que el nivel de privilegio 10 tiene mayor prioridad que 11 (los niveles de privilegio se utilizan en ambientes multiusuario y el sistema indica que normalmente cuando se presenta una violación privilegio).

15	3	2	1	0
SELECTOR		TI	RPL	

Las tablas de descriptores globales y locales se encuentran en el sistema de memoria. Con el fin de acceder y especificar la dirección de estas tablas, los microprocesadores 80286, 80386, 80486, Pentium, Pentium Pro y Pentium II contienen registros invisibles para el programa. Los registros invisibles para el programa no son diseccionados directamente por el software, motivo por el cual reciben este nombre (TR, LDTR,GDTR,IDTR)

La parte invisible de estos registros recibe el nombre de caché (no se debe confundir con los niveles 1 o 2 encontradas en el microprocesador). Cada vez que se cambia el número en el registro de segmento, la parte del registro de segmentos invisible para el programa es cargada en la dirección base., el límite y los derechos de acceso. Cuando se escribe un nuevo número en un registro, el microprocesador accede a la tabla de descriptores y carga el descriptor en la caché invisible para el programa de forma parte del registro del segmento. El descriptor se mantiene ahí y es usado para acceder al segmento de memoria hasta que el número de segmento es cambiado nuevamente. Esto permite al microprocesador acceder repetidamente a un segmento de memoria sin consultar en cada ocasión la tabla de descriptores (de ahí el término caché).

El GDTR (registro de tabla de descriptores globales) y el IDTR (registro de tabla de descriptores de interrupción) contiene la dirección base y límite de la tabla de descriptores. El límite de cada tabla de descriptores es de 16 bits debido a que la máxima longitud de la tabla es de 64 KB. Cuando se desea

la operación en modo protegido, la dirección y el límite de la tabla de descriptores globales son cargados en el GDTR. Antes de usar el modo protegido, debe inicializarse también la tabla de descriptores de interrupción y el IDTR.

La ubicación de la tabla de descriptores locales es seleccionada de la tabla de descriptores globales (uno de los descriptores se configura para direccionar la tabla de descriptores locales).

Para acceder a la tabla de descriptores locales , LDTR (registro de la tabla de descriptores locales) es cargado con un selector, al igual que un registro de segmento es cargado con un selector. Este selector accede a la tabla de descriptores globales y carga la dirección de base, el límite y los derechos de acceso de la tabla de descriptores hacia la parte caché del LDTR.

El TR (registro de tarea) contiene un selector, el cual accede a un descriptor que define una tarea. Una tarea es con frecuencia un procedimiento o un programa de aplicación. El descriptor para el procedimiento o programa es almacenado en la tabla de descriptores globales, de forma que el acceso pueda se controlado por medio de los niveles de privilegio. El registro de tarea permite la conmutación de contexto o de tarea en aprox. 17 us. La conmutación de tareas se efectúa en lapsos razonablemente cortos y permite a los sistemas multitareas cambiar una tarea a otra de manera simple y ordenada.

Base Teórica

Índice

Boot Strap

Boot Strap:

Es el proceso de carga de la PC.
Al encender la computadora (o resetearla), lo primero que realiza el BIOS es el **POST (Power On Self Test)**, el cual consta de inicializar datos (registros).
Luego, se llama a la interrupción 19, la cual lee el sector de booteo de la primera unidad de diskette, de no encontrar un sector de booteo correcto, lee el **MBR (Master Boot Record)** de la primera unidad de disco rígido, allí se aloja un programa, el cual indica las particiones booteables de su tabla de particiones, de donde se intentara leer un sector de booteo correcto.
El sector de booteo de una unidad de diskette, o el MBR de una unidad de disco rígido se busca en el primer sector del primer track de la primer cabeza de la unidad.
Para el BIOS un sector de booteo correcto es aquel que contenga 0x0AA55 a un desplazamiento de 510 bytes.
Si no se encuentra un sector de booteo correcto, se llama la interrupción 18.
Al encontrar un sector de booteo correcto, la interrupción 19 procede a levantar a memoria dicho sector (512 bytes), especificamente en la posición de memoria 0x7C00, y luego se hace un salto incondicional a la misma, y el código del sector de booteo toma el control.
A esta altura todo lo que ha sido inicializado es el área de datos del BIOS (0x40:0x0) y las interrupciones del BIOS (0x10 - 0x1A). El resto de la memoria esta libre, pero no necesariamente seteada a 0x0.
Cabe aclarar que todo esto sucede en modo real.

Loader

Loader:

Como se menciona en esta documentación, el bootstrap es el proceso por medio del cual, se levanta a memoria una porción del Sistema Operativo, la cual es muy pequena, por la limitación dada por el tamano de un sector (512 bytes).
Esta porción del Sistema Operativo tiene una función muy específica, la de levantar un programita a memoria, por medio del cual se procederá a levantar posteriormente el kernel, por lo cual se la denomina LOADER.
A esta altura, la PC sigue funcionando en modo real.
Luego que el LOADER es levantado a memoria, se procede con su ejecución, y finalizada la misma, quedará delegada en el kernel toda la funcionalidad del Sistema Operativo.

El Kernel

El kernel o también conocido como el núcleo del sistema operativo es la parte central del mismo,esta constítuido por un conjunto de rutinas cuya misión es la de gestionar el procesador, la memoria, la entrada/salida y el resto de procesos disponibles en la instalación. Por tal motivo, el núcleo permanece en memoria principal, en tanto que otras partes del sistema operativo son transportadas de un lado a otro entre el almacenamiento primario y el secundario, según las necesidades.

El núcleo es quien inhabilita las interrupciones mientras responde a una de ellas; las interrupciones son habilitadas nuevamente después de completar el proceso de atención.

Un núcleo de sistema operativo contiene normalmente el código necesario para la realización de las funciones siguientes:

- Manipulación de interrupciones.
- Creación y destrucción de procesos.
- Cambio de estados de proceso.
- Despacho.
- Suspensión y reanudación de procesos.
- Sincronización de procesos.
- Comunicación entre procesos.
- Manipulación del bloque de control de proceso.
- Soporte de las actividades de entrada/salida.
- Soporte de la asignación y desasignación del almacenamiento.
- Soporte del sistema de archivos.
- Soporte de un mecanismo de llamada/regreso al procedimiento.
- Soporte de ciertas funciones contables del sistema.

Cabe destacar que podemos clasificar al kernel en dos grandes grupos:

Kernel monolítico
Los núcleos monolíticos generalmente están divididos en dos partes estructuradas: el núcleo dependiente del hardware y el núcleo independiente del hardware. El núcleo dependiente se encarga de manejar las interrupciones del hardware, hacer el manejo de bajo nivel de memoria y discos y trabajar con los manejadores de dispositivos de bajo nivel, principalmente. El núcleo independiente del hardware se encarga de ofrecer las llamadas al sistema, manejar los sistemas de archivos y la planificación de procesos. Para el usuario esta división generalmente pasa desapercibida. Para un mismo sistema operativo corriendo en diferentes plataformas, el núcleo independiente es exactamente el mismo, mientras que el dependiente debe re-escribirse. Este es el núcleo básico de un sistema operativo centralizado.

Microkernel

Un núcleo con 'arquitectura' micronúcleo es aquél que contiene únicamente el manejo de procesos y threads, el de manejo bajo de memoria, da soporte a las comunicaciones y maneja las interrupciones y operaciones de bajo nivel de entrada-salida. [Tan92]. En los sistemas oprativos que cuentan con este tipo de núcleo se usan procesos 'servidores' que se encargan de ofrecer el resto de servicios (por ejemplo el de sistema de archivos) y que utilizan al núcleo a través del soporte de comunicaciones.

Este diseño permite que los servidores no estén atados a un fabricante en especial, incluso el usuario puede escoger o programar sus propios servidores. La mayoría de los sistemas operativos que usan este esquema manejan los recursos de la computadora como si fueran objetos: los servidores ofrecen una serie de 'llamadas' o 'métodos' utilizables con un comportamiento coherente y estructurado. Otra de las características importantes de los micronúcleos es el manejo de threads. Cuando un proceso está formado de un solo thread, éste es un proceso normal como en cualquier sistema operativo.

Los usos más comunes de los micronúcleos es en los sistemas operativos que intentan ser distribuídos, y en aquellos que sirven como base para instalar sobre ellos otros sistemas operativos. Por ejemplo, el sistema operativo AMOEBA intenta ser distribuído y el sistema operativo MACH sirve como base para instalar sobre él DOS, UNIX, etc.

Boot Strap

Administración de memoria en modo protegido

1 - Introducción

En el modo protegido del i386 se cuenta con dos ventajas en la administración de memoria: paginación y segmentación.
La segmentación brinda la posibilidad de "aislar" módulos de memoria (código, datos, pila), para que diferentes tareas puedan correr sobre el mismo procesador si interferir una con otra.
La paginación brinda la posibilidad de tener en memoria pequeñas partes de los módulos en función de lo requerido. La paginación puede utilizarse para brindar "aislamiento" entre diferentes tareas.
Cuando operamos en modo protegido, alguna forma de segmentación debe ser utilizada, debido a que la segmentación no tiene la posibilidad de habilitarla o no, en cambio la paginación si.
Estos dos mecanismos, pueden ser configurados para soportar un único proceso (o tarea), multitarea, o sistemas multiprocesamiento que utilicen memoria compartida.
La segmentación brinda un mecanismo para la división del espacio de memoria direccionable por el procesador (llamado ESPACIO DE DIRECCIONAMIENTO LINEAL) en espacios de direcciones protegidos más pequeños (llamados SEGEMENTOS).

Los segmentos pueden ser usados para almacenar CODIGO, DATOS o PILA, para un proceso o estructuras de datos del sistema (como TSS's o LDT's). Si mas de un proceso corre en un procesador, a cada uno de ellos se le puede asignar su propio juego de segmentos. El procesador fuerza los limites entre estos segmentos, y asegura que un proceso no interfiera con la ejecución del otro escribiendo en los segmentos de memoria que no le fueron asignados.
La segmentación también brinda la posibilidad de tipificar los segmentos, para evitar que ciertas operaciones no se puedan realizar en ciertos tipos de segmentos (por ejemplo escribir en un segmento de código).

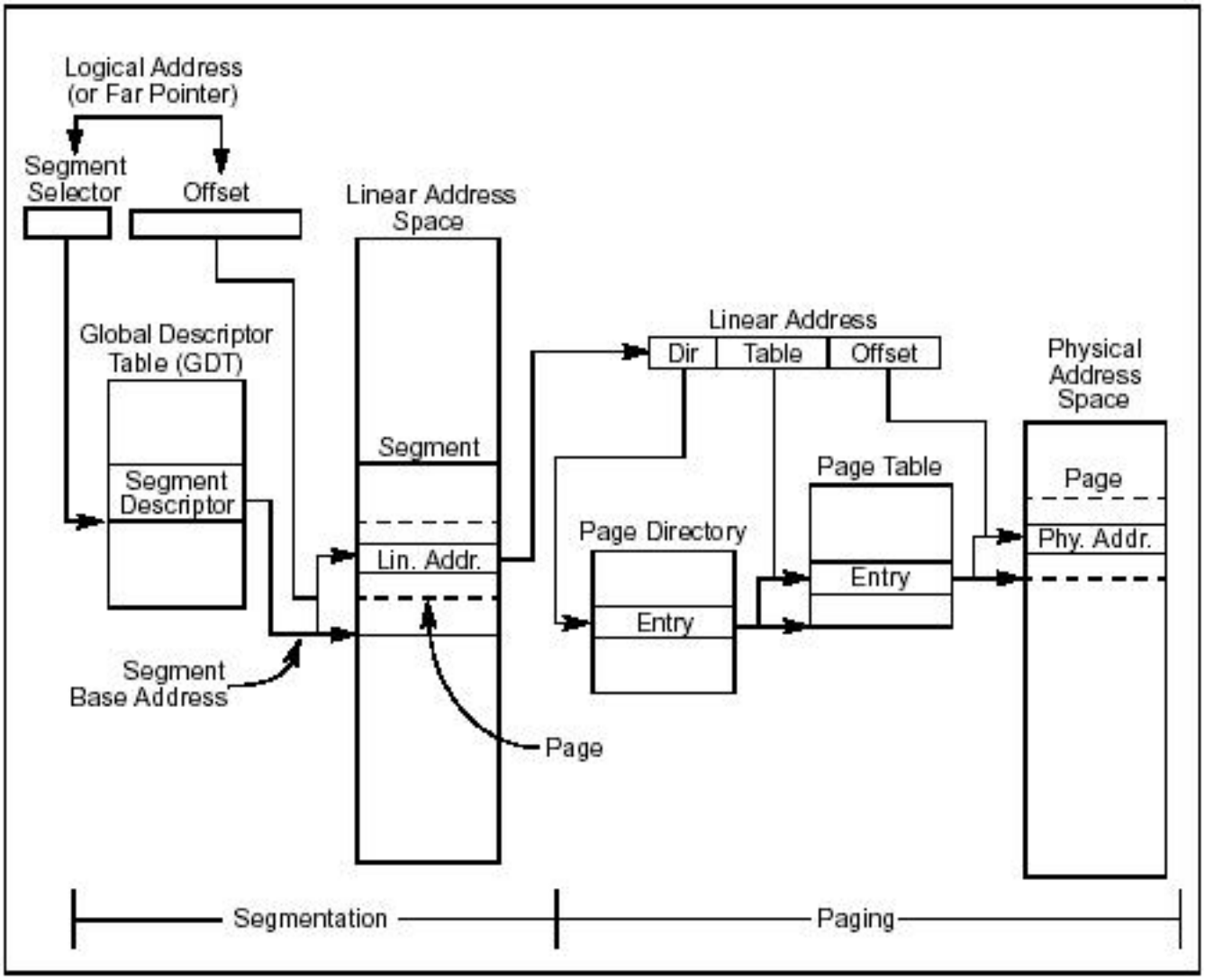
Todos los segmentos de un sistema deben ubicarse dentro del espacio de direccionamiento lineal del procesador. Para ubicar un byte en un segmento en particular, se requiere de una DIRECCION LOGICA (a veces llamada puntero lejano). Una dirección lógica consta de un SELECTOR de segmento y un desplazamiento dentro del mismo. Un selector de segmento es un IDENTIFICADOR UNICO del segmento, el mismo se lo ubica a través de un desplazamiento dentro de la tabla de descriptores de segmentos (por ejemplo la GDT, global descriptor table), en dicha posición se encontrara una estructura del sistema, reconocida por el procesador como descriptor de segmento, conteniendo la BASE del segmento (ubicación del primer byte del segmento dentro del espacio direccionable lineal de memoria), su LIMITE (tamaño del segmento),

sus permisos de acceso y nivel de privilegio, y tipo de segmento.

El desplazamiento, de la dirección lógica, es sumado a la base definida en el descriptor del segmento, el cual es parte de la dirección lógica, formando así la dirección lineal.

Si no se utiliza paginación la dirección lineal es mapeada directamente a la DIRECCION FISICA de memoria. La dirección física de memoria es el rango de direcciones que el procesador es capaz de generar a través de su bus de direcciones.
Comúnmente en un sistema multitarea se define un espacio de direccionamiento lineal mas grande, conviene poder abarcarlo en espacio físico de direccionamiento, para ello es necesario algún método para "virtual izar" el espacio lineal. Esta virtualización es manejada por el mecanismo de paginación del procesador.

La paginación soporta un entorno de "memoria virtual", en donde un gran espacio de memoria lineal es simulado con un pequeño espacio de memoria física (RAM y ROM) y un sector de disco. Cuando se utiliza paginación, los segmentos son divididos en páginas, cada una, comúnmente de 4K, las cuales son almacenadas o en el espacio de memoria física, o en el disco. El sistema operativo mantiene un directorio de páginas, y un juego de tablas de páginas, para tener un seguimiento de las mismas. Cuando un proceso o una tarea intenta acceder a una dirección del espacio de memoria lineal, el procesador consulta el directorio de páginas y las tablas para transformar la dirección lineal en una dirección del espacio físico de memoria, y luego realiza la operación requerida (lectura/escritura) en la ubicación deseada. Si la pagina solicitada no se encuentra en el espacio de direccionamiento físico, el procesador interrumpe la ejecución del programa (generando una EXCEPCION page fault). El sistema operativo obtiene la página del disco y la ubica en el espacio de direccionamiento físico para luego continuar con la ejecución del proceso o tarea interrupida.
Cuando la paginación es implementada correctamente en el sistema operativo, el swapping de páginas entre el espacio físico de direccionamiento y el disco, es transparente para el correcto funcionamiento del proceso o tarea. Hasta los procesos escritos para las arquitecturas de Intel de 16 bits, pueden ser paginados (transparentemente) cuando corren en el modo virtual-8086.



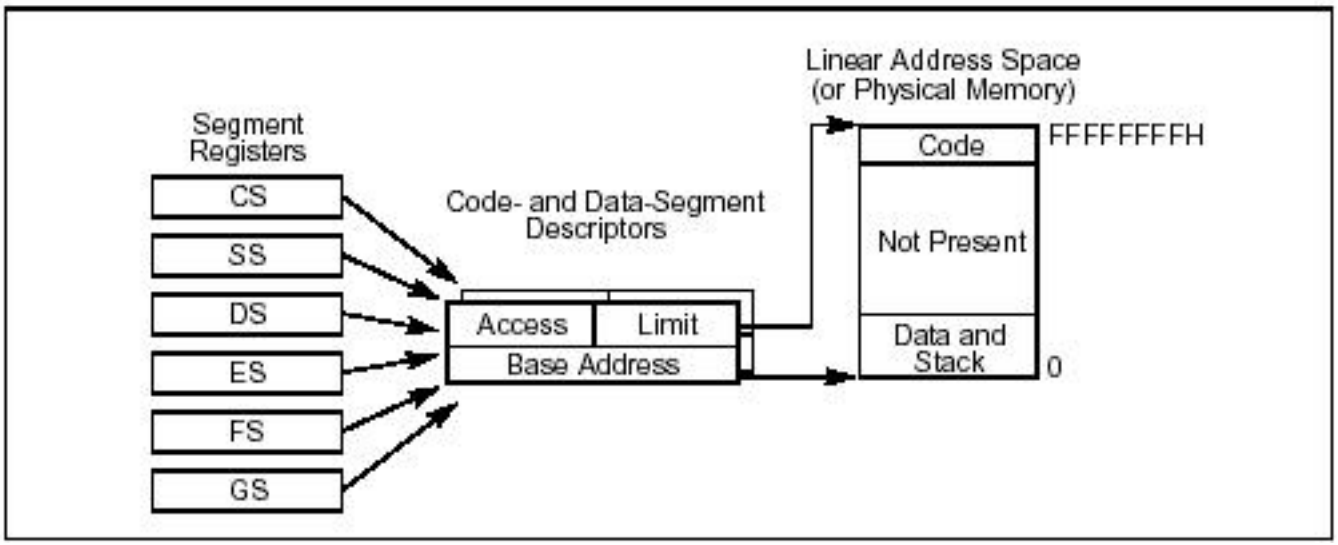
2 - Formas de segmentación

El mecanismo de segmentación de la arquitectura Intel puede ser utilizado para implementar una amplia variedad de diseños de sistemas. Estos disnos varían desde el modelo Flat, el cual utiliza minimamente la segmentación para proteger los procesos, hasta el modelo multisegmentado el cual implementa la segmentación para crear un entorno de operación robusto en el cual múltiples procesos y tareas pueden ser ejecutados con seguridad.

2.1 - Segmentación Flat

Este el modelo mas sencillo de segmentación. En el mismo, el sistema operativo y las aplicaciones, tienen acceso a un espacio de direccionamiento continuo y no segmentado. Lo mas extenso como sea posible, ocultando el mecanismo de segmentación de la arquitectura, del diseñador del sistema y del programador de aplicaciones. Para implementar este modelo es necesario crear al menos dos descriptores de segmentos, uno para referenciar a un segmento de código y otro para referenciar a uno de datos, aunque este mapeados a todo el espacio de direccionamiento lineal del sistema: esto es, ambos segmentos tienen base 0x0, e igual limite:

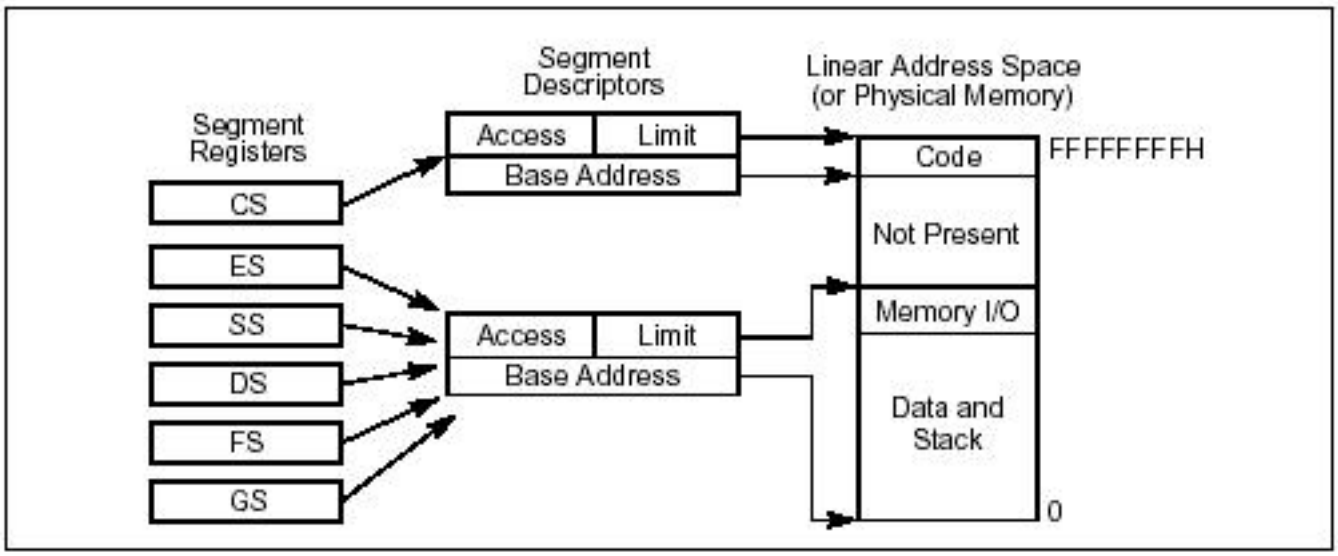
4GB. Configurando el limite en 4GB, el mecanismo de segmentación estará liberado de producir excepciones por referencias a memoria fuera de limite (ya que 4GB es el máximo direccionable por la arquitectura del i386), aunque no resida memoria física en una dirección particular. ROM (EPROM) generalmente se ubica en la parte superior del espacio de direccionamiento físico, porque el procesador comienza a ejecutar en FFFF_FFF0H. RAM (DRAM) esta ubicada en la parte inferior del espacio de direccionamiento físico, ya que al iniciar el registro de segmento de datos se encuentra con valor 0H.



2.2 - Segmentación Flat Protegida

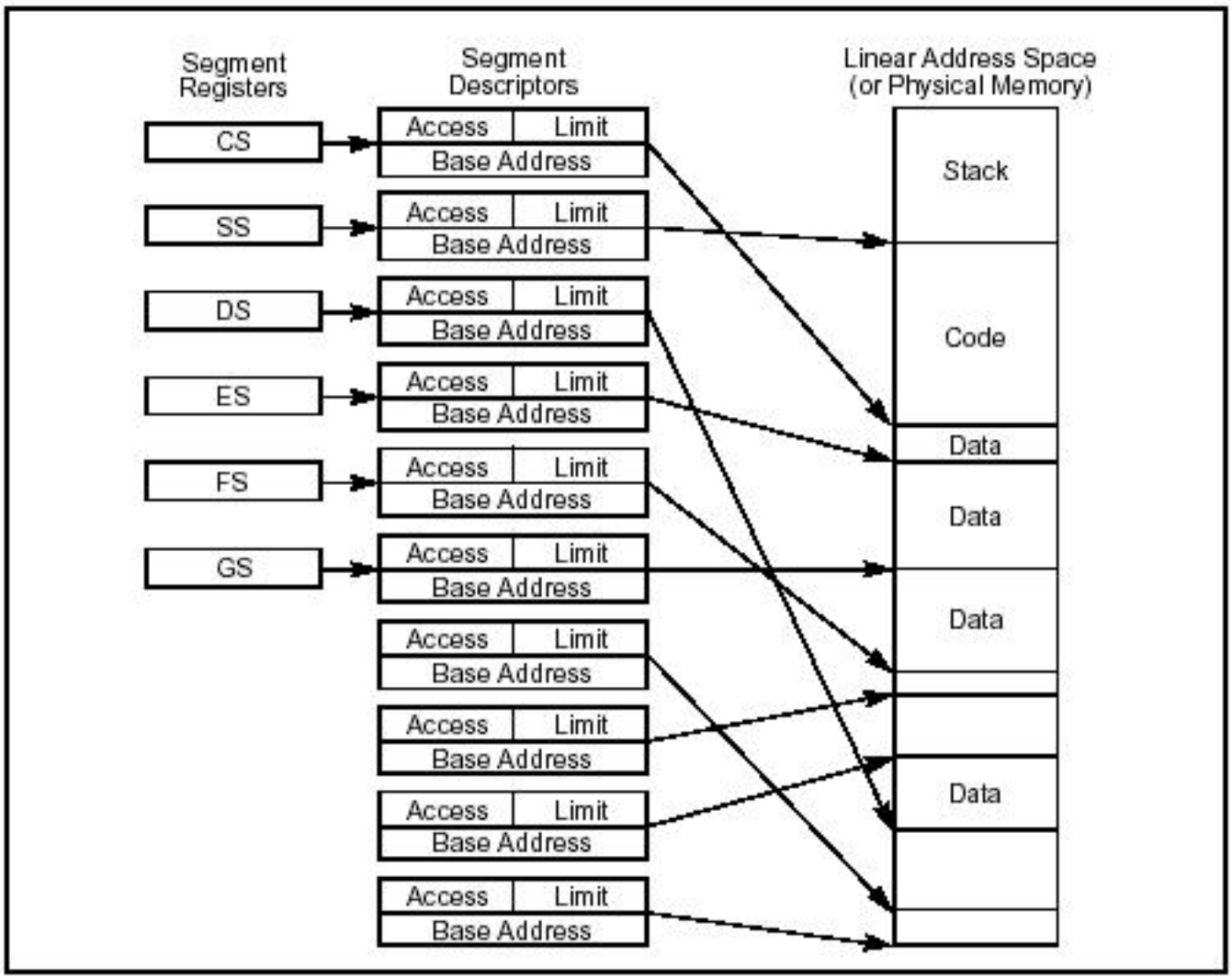
Este modelo es similar al modelo Flat básico, excepto que los límites de los segmentos son configurados para incluir solo el rango de direcciones para el cual existe memoria física actualmente. Una excepción de protección-general (#GP) es generada ante cualquier intento de acceder a memoria no existente, lo cual no ocurre en el modelo anterior. Este modelo provee un nivel mínimo de protección por hardware contra algunos tipos de errores de programación.

Se puede incrementar la complejidad del modelo para proveer mayor protección. Por ejemplo, en el mecanismo de paginación, para proveer "aislamiento" entre código y datos del usuario y del supervisor, es necesario definir cuatro segmentos: uno de código y otro de datos para el nivel de privilegio 3 para el usuario, y otros dos para el nivel de privilegio 0 para el supervisor. Usualmente estos segmentos se encuentran solapados y tienen una base común 0H. Este modelo Flat junto con una estructura de paginación puede proteger el sistema operativo de las aplicaciones, y agregando una estructura de paginación separada para cada proceso o tarea, puede, a su vez, proteger a las aplicaciones entre ellas. Modelos similares son utilizados por varios sistemas operativos multitarea populares.



2.3 - Segmentación Multisegmentos

Un modelo multisegmentos usa todas las capacidades del mecanismo de segmentación para proveer una forzosa protección por hardware del código, estructura de datos, y procesos y tareas. En este modelo, a cada proceso o tarea, se le asigna su propia tabla de descriptores de segmentos y sus propios segmentos. Los segmentos pueden ser completamente privados al proceso o tarea al que fueron asignados, o compartidos con otros procesos o tareas. El acceso a todos los segmentos y a los entornos de ejecución de cada proceso o tarea individual corriendo en el sistema, es controlada por el hardware.
Se puede utilizar control de acceso no solo para proteger contra referencias a memoria fuera del límite del segmento, sino contra la ejecución de operaciones no permitidas en ciertos segmentos, por ejemplo, el hardware no permitiría que se escriba en un segmento de solo lectura como lo son los de código. La información de permisos de acceso creada para el segmento puede ser utilizada para la creación de ANILLOS O NIVELES DE PROTECCION. Los niveles de protección pueden ser utilizados para proteger los procedimientos del sistema operativo de accesos no autorizados de programas de aplicación.



3 - Segmentación y Paginación

Aquí la segmentación es usada para definir particiones lógicas de memoria orientadas al control de acceso y la paginación es usada para manejar la localización de la memoria dentro de las particiones. El sistema MULTICS resolvió el problema de la fragmentación externa mediante la "paginación de los segmentos". * La solución difiere de la segmentación pura en que la entrada de la tabla de segmentos no contiene la dirección base del segmento, sino la dirección base de una tabla de páginas para ese segmento. Entonces podemos decir que gracias a la paginación se elimina fragmentación externa y se provecha eficientemente la memoria y gracias a la segmentación, es más visible al programador y da soporte para compartición y protección.
Dirección lógica: Desde el punto de vista del usuario esta formada por:
Nro de segmento | desplazamiento

Desde el punto de vista de la máquina esta formada por:
Nro segmento | Nro página | desplazamiento

Estructuras de datos utilizadas:
-Cada proceso

-Una tabla de segmentos
-Una tabla de páginas por segmento

Manual Técnico

Lenguaje Ensamblador

El lenguaje ensamblador es código máquina puro. Al programar con este herramienta se le dan instrucciones directamente al microprocesador de la CPU. O sea que con esto podemos hacer que la computadora haga lo que nosotros querramos. Es muy importante, saber que estamos haciendo con cada instrucción del lenguaje ensamblador. Antes de programar directamente en assembler se debe tener conocimientos básicos del funcionamiento interno de la computadora y de la manera en que esta maneja la información internamente. Los conocimientos básicos se dan en la Introducción de este documento.

Introducción

1.1 Sistema Numérico

La computadora No almacena los datos en memoria en números decimales (base 10). Porque se hace mucho má,s fácil el Hardware, s se almacenan los datos en formato binario (base 2). Primero veremos una breve vista del sistema Decimal.

1.1.1 Decimal Los números en base 10 estan compuestos de 10 posibles dígitos (0-9). Cada dígito es un múmero que tiene una potencia de 10 asociada con esta bese según la posición del múmero. Por ejemplo:

```
234 = 2 * 10^2 + 3 * 10^1 + 4 * 10^0 = 2 * 100 + 3 * 10 + 4 * 1
```

1.1.2 Binario Los números en base dos están compuestos por dos dígitos (0 and 1). Cada dígito de un número tiene una potencia de la base asociada a la posición del númnero. (Un simple dígito binario se lo llama una BIT) Por ejemplo:

```
11001 en base 2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0
                        = 16 + 8 + 1 = 25
```

Esto muestra como un número binario se convierte en decimal. La Tabla muestra como los primeros números binarios son convertidos. Aqui hay un ejemplo:

Decimal	Binario	Decimal	Binario
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Tabla 1.1: Números decimales de 0 a 15 convertidos a Binario

La figura 1.1 muestra como individuales dígitos binarios (bits) se suman:

Sin Acarreo(Carry) previo

```
  0 0 1 1
+
  0 1 0 1
---------
  0 1 1 0
    c
```

Con Acarreo(Carry) previo

```
  0 0 1 1
+
  0 1 0 1
---------
  1 0 0 1
  c c c
```

Figura 1.1: Suma de binarios (c = carry)

```
 11011
+10001
------
101100
```

Considere la siguiente división binaria:

```
1101 ÷ 10 = 110 r 1
```

Esto muestra que dividiendo por dos en binario se corren todos los bits a la derecha una posición y se mueve el orinal bit de más a la derecha al resto. (Analogamente, dividiendo por diez en decimal se corren todos los dígitos decimales a la derecha un lugar y se mueve el original dígito al resto.) De esta manera, se puede convertir un número decimal en su equivalente binario.

Decimal	Binario
25 ÷ 2 = 12 r **1**	11001 ÷ 10 = 1100 r **1**
12 ÷ 2 = 6 r **0**	1100 ÷ 10 = 110 r **0**

6 ÷ 2 = 3 r **0** 110 ÷ 10 = 11 r **0**
3 ÷ 2 = 1 r **1** 11 ÷ 10 = 1 r **1**
1 ÷ 2 = 0 r **1** 1 ÷ 10 = 0 r **1**

Este método encuentra primero al dígito que está más a la derecha, o sea el bit menos significativo. El dígito de más a la izquierda es llamado el bit más significativo. Una unidad básica de memoria tiene 8 bits y se llama *byte*.

1.1.3 Hexadecimal Los números hexadecimales usa base 16. Hexadecimal (o hex abreviado) se puede usar como una forma más corta de escribir números binarios. Hex tiene 16 símbolos. En ellos hay un problema que es, que símbolos usamos para representar los dígitos que viene luego del 9, entonces la solución es usar letras. En resumen, los 16 símbolos son: los símbolos de 0 al 9 y luego A, B, C, D, E y F. El símbolo A es el equivalente al 10, el B al 11, etc. Cada dígito es un número hexa y tiene una potencia de 16 asociada. Por ejemplo:

```
2BD16= 2 * 16^2 + 11 * 16^1 + 13 * 16^0
     = 512 + 176 + 13
     = 701
```

Para covertir de decimal a hexa, se usa la idea que se usa para conversión a binario exceptuando el dividir por 16. A continuación un ejemplo:

589 ÷ 16 = 36 r **13**
36 ÷ 16 = 2 r **4**
2 ÷ 16 = 0 r **2**

Por lo tanto, 589 = 24D(base 16). La razón que hexa es muy poderoso es que hay una manera muy simple de convertir de hexa a binario y viseversa. Hexa provee un manera mucho más compacta que la forma en binario. Para convertir un hexa a binario, simplemente hay que convertir cada dígito hexa en 4-bit números binarios. Por ejemplo, 24D(base 16) se convierte a 0010 0100 1101(base 2). Note que el cero principal de los 4-bits es importante!
Convertir de binario a hexa es muy fácil. Simplemente haga la conversión inversa. Convierta cada 4-bits del binario a hexa. Recuerde empezar del extremo derecho y no del extremo izquierdo del número binario. Ejemplo:

```
110 0000 0101 1010 0111 1110 (binario     -- base 2)
 6    0    5    A    7    E  (hexadecimal -- base 16)
```

Un número de 4-bit se llama *nibble*. Y así cada dígito del hexa corresponde a un *nibble*. Dos nibbles hacen un byte y por lo tanto, un byte puede representarse por un número hexa de 2 dígitos. El valor de un byte va de 0 a 11111111 en el binario, 0 a FF en hexa y 0 a 255 en el decimal.

1.2 Organización de la computadora

1.2.1 Memoria
La unidad básica de memoria es el byte. Una computadora con 32 Meg de RAM puede almacenar 32 millones de bytes de información. Cada byte en la memoria está etiquetado por un único número conocido como su dirección. Un ejemplo se muestra a continuación:

Dirección	0	1	2	3	4	5	6	7
Memoria	2A	45	B8	20	8F	CD	12	2E

Direcciones de Memoria

Todo dato en la memoria es numérico. Los cáracteres son guardados usando un código de cáracteres. La PC usa el código del cáracter más común, conocido como ASCII (American Standard Code for Information Interchange -- Código Estandar Americano para el Intercambio de Información). A menudo la memoria se usa "pedazos" más grandes que los solos bytes.

¿Usted necesita assembler?

Si bien, no querríamos interferir con lo que está haciendo, pero le vamos a dar algunos consejos de los duramente ganamos con la experiencia.

2.1. Pros y contras del Lenguaje Ensamblador

2.1.1. Las ventajas del Lenguaje Ensamblador

El lenguaje assembler puede expresar cosas muy de bajo nivel.

- Puede acceder a los registros dependientes de la máquina y E/S
- Puede controlar la conducta exacta del código en secciones críticas que podrían involucrar abrazo mortal (deadlock) entre multiples hilos de software o dispositivos de hardware.
- Puede romper las convenciones de su compilador usual y realizar algunas optimizaciones (como romper temporalmente romper las reglas sobre la asignación de memoria, manejo de hilos, convenciones de llamadas, etc)
- Puede construir interfaces entre fragmentos de código que usan convenciones incompatibles (ej. código produdido por diferentes compiladores, o separar una interface de bajo nivel)
- Puede hacer el acceso a modos de la programación no usuales en su procesador (por ejemplo, modo de 16 bits para interfases de startup, firmware, o código legal en computadoras Intel)
- Puede producir el código bastante rápido para blucles que no se logra con un compilador no perfeccionando o malo
- Puede producir código perfeccionado a mano y puesto a punto para su configuración de hardware particular perfectamente, aunque no le sirva a nadie más.

- Puede escribir algún código para su nuevo lenguaje de compilador optimizado (esto es algo que pocas capos alguna vez harán, e incluso ellos no lo hacen a menudo).

2.1.2. Las desventajas del Lenguaje Ensamblador

- El lenguaje *assembler* es un lenguaje de muy bajo nivel (el más bajo antes de codificar a mano modelos de instrucciones binarias). Esto significa que es difícil y tedioso de escribir inicialmente.
- Sus errores pueden ser muy difíciles de identificar.
- Su código puede ser bastante difícil de entender y modificar, es decir de mantener.
- El resultado es no portátil a otras arquitecturas, mientras que si existe compatibilidad, su código, sólo se perfeccionará para una cierta aplicación de una misma arquitectura: por ejemplo, entre las plataformas compatibles Intel cada CPU tiene su diseño y sus variaciones (la latencia relativa, el rendimiento, su capacidad de: unidades de proceso, caches, Memoria RAM, buses, discos, presencia de FPU, MMX, 3DNOW, las extensiones de SIMD, etc) implican técnicas de optimización potencialmente y completamente diferentes. Los diseños de CPU hoy incluyen: Intel 386, 486, el Pentium, PPro, PII, PIII, PIV,,; CYRIX 5X86, 6X86, M2,; AMD K5, K6 (K6-2, K6-III), K7 (Athlon, Duron). los Nuevos diseños siguen incrementandose, y no dan tiempo para que su código sea moderno.
- Se gasta mucho tiempo en detalles y no se puede enfocar al plan algorítmico del problema, o para ganar la parte para ganar velocidad (por ejemplo: se gasta tiempo construyendo algo primitivo como manipulación de Arreglos y Listas en *assembler*; que se tiene muchas facilides de contrucción y uso en leguajes de alto nivel)
- Un pequeño cambio en el diseño de un algoritmo puede producir la completa invalidación de todo el código *assembler* existente. Por lo que se vería obligado a escribir todo el código nuevamente por cada vez que realice un pequeño cambio al diseño de un algoritmo.

Arquitectura del Microprocesador

Es necesario conocer la arquitectura interna visible para el programa, correspondiente a los microprocesadores 8086 al pentiun II. También veremos las funciones y objetivos de estos registros internos.

El modelo de programación de los microprocesadores 8086 al Pentium II se considera **visible para el programa**, ya que sus registros son utilizados durante la programación de aplicaciones y son especificados por las instrucciones. Otros registros se consideran como **invisibles para el programa** debido a que no están accesibles directamente durante la programación de las aplicaciones, aunque pueden ser usados indirectamente durante el programación del sistema. **únicamente el 80286 y posteriores contienen registros invisibles para el programa que se utilizan para controlar y operar el sistema de memoria en modo protegido** .

El modelo de programación de los microprocesadores del 8086 al Pentium II. (Los antiguos 8086,8088 y 80286 poseen arquitecturas internas de 16 bits, un subconjunto de los registros mostrados en la figura. Los microprocesadores 80386,80486, Pentium, Pentium Pro y Pentium II poseen arquitecturas internas completas de 32 bits. Las arquitecturas de los antiguos 8086 al 80286 son totalmente compatibles con los 80386 y Pentum II.

El modelo de programación contiene registros de 8,16 y 32 bits. Los registros de 8 bits son AH, AL, BH, BL,CH,CL,DH y DL y son especificados cuando una instrucción se forma utilizando estas denominaciones de dos letras. Los registros de 16 bits son AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS y GS. Estos registros también se especifican con denominaciones de dos letras. Los registros extendidos de 32 bits son EAX, EBX,ECX,EDX,ESP,EBP,EDI,ESI,EIP, y EFLAGS. Los registros extendidos de 32 bits, y los de 16 bits FS y GS, están disponibles solamente en los microprocesadores 80386 y posteriores.

Regitros de propósito múltiples

EAX (acumulador): EAX es especificado como un registro de 32 bits, como un registro de 16 bits AX o como un de 8 bits AH o AL. El acumulador se utiliza para instrucciones tales como la división, la multiplicación y algunas instrucciones de formato. En los microprocesadores 80386 y posteriores, el registro EAX puede contener también la direccioón de desplazamiento de una localidad en el sistema de memoria.

EBX (índice de base):El EBX puede ser direccionado como EBX,BX,BH o BL. El registro BX en ocaciones contiene la dirección de desplazamiento de una localidad en el sistema de memoria para todas las versiones de microprocesadores. En los 80386 y posteriores, EBX también puede direccionar datos de memoria.

ECX (cuenta): El ECX es un registro de próposito general que también contiene el contador para las

distintas instrucciones. En los 80386 y posteriores, el registro ECX también puede contener la dirección de desplazamiento de los datos en memoria. Las instrucciones que usan un contador son las instrucciones de cadena repetidas, las instrucciones de desplazamiento , rotacion y LOOP/LOOPD. Las instrucciones de desplazamiento y rotación utilizan al CL como contador, las instrucciones de cadenas repetidas usan a CX y las instrucciones LOOP/LOOPD, usan ya sea a CX o a ECX.

EDX (datos): es un registro de próposito general que contiene parte del producto de una multiplicación o parte del dividendo de una división. En el 80386 y posteriores este registro también puede direccionar datos en memoria.

EBP (apuntador de base): EBP apunta hacia una localidad de memoria, para la transferencias de datos de memoria, en todas las versiones del microprocesador. Este registro se direcciona ya sea como BP o como EBP.

EDI (índice de destino): A menudo, EDI direcciona datos del destino de las cadenas para las instrucciones de cadena. También funciona como registro de propósito general tanto de 32 bits (EDI), como de 16 (DI).

ESI (índice de fuente): El ESI puede utilizarse ya sea como ESI o como SI. El registro del índice fuente con frecuencia direcciona datos del origen de las cadenas para las intrucciones de cadena. Al igual que EDI, ESI también funciona como un registro de propósito general. Como registro de 16 bits, se direcciona como SI; como registro de 32 bits, se direcciona como ESI.

Registros de segmento

Los registros de segmento son registros adicionales que generan direcciones de memoria al combinarse con otros registros del microprocesador. Existen ya sea cuatro o seis registros de segmento en diferentes versiones del microprocesador. Un registro de segmento funciona de manera diferente dependiendo de si el microprocesador opera en modo real o en modo protegido. A continuación encontrará una lista de cada registro de segmento, con su función en el sistema:

CS (código): El segmento de código es una sección de la memoria que contiene el código (programas y procedimientos) utilizados por el microprocesador. El registro del segmento de código define la dirección inicial de la sección de memoria que contiene el código. En operación de modo real, define el principio de una sección de 64 KB de memoria; en modo protegido, selecciona un descriptor que muestra la dirección de inicio y la longitud de una sección de memoria que contiene código. El segmento de código está limitado a 64 Kb en los 8088 al 80286 y a 4 GB en los 80386 y posteriores, cuando estos microprocesadores operan en modo protegido.

DS (datos): el segmento de datos es una sección de memoria que contiene la mayoría de los datos utilizados por un programa. En este segmento, se accede a los datos por medio de un desplazamiento o a través del contenido de otros registros que contienen la dirección de desplazamiento. Al igual que los segmentos de código y otros segmentos, su longitud está limitada a 64 Kb en los 8086 al 80286 y a 4 GB en los 80386 y posteriores.

ES (extra): el segmento extra es un segmento de datos adicional utilizado por algunas instrucciones de cadena para constituir el destino de los datos.

SS (pila): el segmento de pila define el área de memoria utilizada para la pila. El punto de entrada de la pila está determinado por los registros de este segemento, así como por los registros de los apuntadores de pila. El registro BP también direcciona datos dentro del segmento de pila.

FS y GS: los segmentos FS y GS son los registros de segmento complementarios que están disponibles en los microprocesadores 80386,80486, Pentium y Pentium Pro y que permiten que los programas accedan a dos segmentos de memoria adicionales.

Puntero de instrucciones

El puntero de instrucciones es un registro de 32 bits llamado EIP, el cual mantiene el offset de la próxima instrucción a ejecutar. El offset siempre es relativo a la base del segmento de código (CS). Los 16 bits menos significativos de EIP conforman el puntero de instrucciones de 16 bits llamado IP, que se utiliza para direccionamiento de 16 bits.

Registros de direcciones del sistema

Existen cuatro registros de direcciones del sistema, pensados para refereciar las tablas o segmentos soportados por el modelo de protección de la CPU 286 y 386 DX. Estas Tablas o Segmentos son:

GDT (Global Descriptor Table)
IDT (Interrupt Descriptor Table)
LDT (Local Descriptor Table)
TSS (Task State Segment)

Las direcciones de estas tablas y segmentos se encuentran en registros especiales, Direcciones del Sistema y Registros de Segmentos del Sistema, los mismos son: GDTR, IDTR, LDTR y TR, respectivamente.

GDTR e IDTR: estos registros almacenan la dirección lineal de la base en 32 Bits y el límite en 16 Bits. Ambos datos referidos a la tabla de descriptores globales y a la tabla de descriptores de interrupciones, respectivamente. Como ambos registros se refieren a un segmento global a todas las tareas en el sistema, deben referenciarse a travéz de una dirección base lineal de 32 Bits y un límite de 16 Bits (sujetos a la traducción necesaria si la paginaci&ocute;n se encuentra activada).

LDTR y TR: estos registros almacenan un selector de 16 Bits, referido a un descriptor de la tabla de descriptores locales o a uno del segmento de estado de tareas, respectivamente. Como ambos se refieren a una tarea en particular, se los puede referenciar a travéz de un selector almacenado en un registro de segmentos del sistema. Notese que existe un descriptor de segmento oculto (invisible para el programador) relacionado con cada registro de segmentos del sistema, el cuál se carga al asignarle un selector al registro.

TSS (Task-State Segment):
Una tarea posee dos elementos básicos, un espacio de ejecución y una TSS. El espacio de ejecución está compuesto por un segmento de código, un segmento para su stack o pila y uno o más segmentos de datos. En general si usa un mecanismo de potección para distintos niveles de privilegios entre las tareas se le da un stack o pila a cada una de estas.
Por otro lado se encuentra la TSS que es una tabla en memoria que almacena el entorno de ejecución o el estado de la tarea.
La TSS tiene la siguiente estructura:

32-Bit Task-State Segment (TSS)

31 15 0	
I/O Map Base Address	T 100
LDT Segment Selector	96
GS	92
FS	88
DS	84
SS	80
CS	76
ES	72
EDI	68
ESI	64
EBP	60
ESP	56
EBX	52
EDX	48
ECX	44
EAX	40
EFLAGS	36
EIP	32
CR3 (PDBR)	28
SS2	24
ESP2	20
SS1	16
ESP1	12
SS0	8
ESP0	4

	Previous Task Link	0

Bits Reservados. Setearlos a 0.

Cada TSS posee uno o varios descriptores de la misma que se encuentra en la GDT. También puede posee un Task-Gate Descriptor. La TSS debe ser única para cada proceso. El micro no permite recursividad de una tarea.

TSS 16-BITS

15	0
Task LDT Selector	42
DS Selector	40
SS Selector	38
CS Selector	36
ES Selector	34
DI	32
SI	30
BP	28
SP	26
BX	24
DX	22
CX	20
AX	18
Flag Word	16
IP (Entry Point)	14
SS2	12
SP2	10
SS1	8
SP1	6
SS0	4
SP0	2
Previous Task Link	0

Registros de Control

En el 80386 aparece una nueva serie de registros, no encontrados anteriormente en los

microprocesadores Intel, como son los registros para control, depuración y verificación. Los registros de control CR0-CR3 controlan varias características, DR0-DR7 facilitan la depuraci&ocute, y los registros TR6 y TR7 se utilizan para verificar la paginación y el uso de la memoria caché.
El registro de control 0 (CR0) es igual al MSW (palabra de estado de máquina) encontrado en el microprocesador 80286, excepto que es de 32 bits, en lugar de 16. El CR1,CR2 y CR3 son registros de control adicionales. El registro de control CR1 no se utiliza en el 80386, pero está reservado para productos posteriores. El registro de control CR2 contiene la dirección lineal de página accedida antes de que ocurriera una interrupción por falla de página. Por uacute;ltimo, el registro de control CR3 contiene la dirección base del directorio de tablas de páginas.Los 12 bits del extremo derecho de la dirección de tabla de páginas de 32 bits contiene ceros y concatenan con el restro del registro para ubicar el inicio de la tabla de páginas de 4 KB de longitud.
El registro CR0 contiene un número de bits de control especiales definidos en el 80386 como sigue:
PG : selecciona la traducción por tabla de páginas de la dirección lineal a una dirección física cuando PG=1. La traducción por tabla de páginas permite asignar cualquier localidad de memoria física a cualquier dirección lineal.
ET: selecciona el coprocesador 80287 cuando ET=0 y al coprocesador 80387 cuando ET=1. TS: indica que el 80386 ha conmutado tareas (en el modo protegido, el cambio del contenido del TR coloca un 1 en TS). Si TS=1, una instrucción del coprocesador numérico ocasiona la interrupción tipo 7 (coprocesador no disponible).
EM: es establecido para ocacionar una interrupción tipo 7 para cada instrucción ESC. Frecuentemente utilizamos esta interrupción para emular, con software, la función del coprocesador. La emulación reduce el costo del sistema, pero frecuentemente requiere por lo menos 100 veces más tiempo para ejecutar las instrucciones emuladas del coprocesador.
MP: es establecido para indicar que el coprocesador aritmético está instalado en el sistema.
PE: es establecido para seleccionar el modo de operación protegido del 80386. Puede también limpiarse para reingresar al modo real. Este bit solamente puede establecese en el 80286. El 80286 no podía regresar al modo real sin una instrucción por hardware, lo que imposibilita su uso en la mayoría de los sistemas que emplean en modo protegido.

Registros de depuración y verificación

Los primeros 4 registros de depuración contienen direcciones lineales de 32 bits de puntos de detención (una dirección lineal es una dirección de 32 bits generada por una instrucción del microprocesador que puede, o no, ser igual a la dirección física). Las direcciones de punto de detección, que pueden ubicar una instrucción o un dato, son constantemente comparadas con las direcciones de punto de detección, que pueden ubicar una instrucción o un dato, son constantemente comparadas con las direcciones que genera el programa. Si ocurre coincidencia, el 80386 ocasionará una interrupción tipo 1 (TRAP o interrupción de depuración), si es instruido por los registros de depuración DR6 y DR7. Esta característica es una versión muy ampliada de la trampa o trazado básico permitido por los microprocesadores anteriore a Intel, por medio de la interrupción tipo 1. Las direcciones de punto de detección son muy útiles para la depuración de software defectuoso. Los bits de control DR6 y DR7 son definidos como sigue:
BT: si está establecido(1), la interrupción de depuración fue ocasionada por una conmutación de tarea.

BS: si está establecido(1), la interrupción de depuración fue ocasionada por el bit TF en el registro de banderas.
BD: si está establecido, la interrupción de depuración fue ocasionada por un intento de lectura del registro de depuración con el bit GD establecido. El bit GD protege a los registros de depuración contra acceso.
B3-B0: indican cuál de las cuatro direcciones de punto de detección ocasionó la interrupción de depuración.
LEN: cada uno de los cuatro campos de longitud pertenece a cada una de las cuatro direcciones de punto de detección almacenadas en DR0-DR3. Estos bits definen el tamaño del acceso en la direccioón del punto de detección como 00 (byte), 01 (palabra) u 11(doble palabra).
RW: cada uno de los cuatro campos de lectura/escritura pertenece a cada una de las 4 direcciones de punto de detección almacenadas en DR0-DR3. El campo RW selecciona la causa de la acción que habilitó una dirección de punto de detección como 00 (acceso a instrucción), 01 (escritura de dato) y 11 (escritura y lectura de dato).
GD: si está establecido, GD evita cualquier lectura o escritura de un registro de depuración, generando la interrupción de depuración. Este bit se limpia automáticamente durante la interrupción de depuración, para que los registros de depuración puedan leerse o cambiarse, si fuera necesario.
GE: si está establecido, selecciona una dirección global de punto de detección para cualquiera de los cuatro registros de dirección, para que los registros de dirección de punto de detención.
LE: si está establecido, selecciona una dirección local de punto de detección para cualquiera de los cuatro registros de dirección, para que los registros de dirección de punto de detención.

Los registros de verificación, TR6 y TR7, se utilizan para examinar el **buffer de traducción de página (TLB)**. El TLB se utiliza con la unidad de paginación del 80386. El TLB contiene las traducciones de direcciones de páginas utilizadas con mayor frecuencia. El TLB contiene las traducciones de direcciones de páginas utilizadas con mayor frecuencia. El TLB reduce el número de lecturas de memoria requeridas para localizar direcciones de páginas traducidas en las tablas de traducción de páginass. El TLB contiene las 32 entradas más comunes de la tabla de páginas, y es verificado con los registros de verificación TR6 y TR7. El regitro TR6 contiene el campo de etiqueta (dirección lineal) del TLB y TR7 contiene la dirección física del TLB.
Para escribir a un elemento del TLB, siga los pasos descritos a continuación:
-Escriba TR7 para la dirección física deseada,PL y los valores de REP.
-Escriba TR6 con la dirección lineal, asegurándose de que C=0.
Para leer un elemento del TLB:
-Escriba TR6 con la dirección lineal,asegurándose que C=1. -Lea tanto a TR6, como a TR7. Si el bit PL indica una coincidencia, entonces los valores deseados de TR6 y TR7 indican el contenido del TLB.

Los bits encontrados en TR6 y TR7 indican las siguientes condiciones:
V Muestra que el elemento en el TLB es válido.
D Indica que el elemento del TLB es inválido o está modificado.
U Un bit para le TLB.
W Indica que el área direccionada por el elemento del TLB puede ser escrita.

C Selecciona una escritura (0) o consulta inmediata (1) para el TLB.
PL Indica una coincidencia si tiene el valor 1 lógico.
REP Selecciona cuál bloque del TLB es escrito.

Compatibilidad

Algunos bits de ciertos registros del 80386 no están definidos. Cuando se obtienen estos bits, deben tratarse como completamente indefinidos. Esto es esencial para la compatibilidad con procesadores futuros. Para ello deben seguirse las siguientes recomendaciones:
- No depender de los estados de cualquiera de los bits no definidos. Estos deben ser enmascarados (mediante la instrucción AND con el bit a enmascarar a cero) cuando se utilizan los registros.
- No depender de los estados de cualquiera de los bits no definidos cuando se los almacena en memoria u otro registro.
- No depender de la habilidad que tiene el procesador de retener información escrita en bits marcados como indefinidos.
- Cuando se cargan registros siempre se deben poner los bits indefinidos a cero.
- Los registros que se almacenaron previamente pueden ser recargados sin necesidad de enmascarar los bits indefinidos.

Los programas que no cumplen con estas indicaciones, pueden llegar a no funcionar en 80486, Pentium y siguientes procesadores, donde los bits no definidos del 80386 poseen algún significado en los otros procesadores.

Licencia GNU

GNU

Las licencias que cubren la mayor parte del software están pensadas para quitarle a usted la libertad de compartirlo y modificarlo. Por el contrario, la Licencia Pública General de GNU pretende garantizarle la libertad de compartir y modificar software libre, para asegurar que el software es libre para todos sus usuarios. Esta Licencia Pública General se aplica a la mayor parte del software del la Free Software Foundation y a cualquier otro programa si sus autores se comprometen a utilizarla. (Existe otro software de la Free Software Foundation que está cubierto por la Licencia Pública General de GNU para Bibliotecas).
Si quiere, también puede aplicarla a sus propios programas. Cuando hablamos de software libre, estamos refiriéndonos a libertad, no a precio.
Nuestras Licencias Públicas Generales están pensadas para asegurarnos de que tenga la libertad de distribuir copias de software libre (y cobrar por ese servicio si quiere), de que reciba el código fuente o que pueda conseguirlo si lo quiere, de que pueda modificar el software o usar fragmentos de él en nuevos programas libres, y de que sepa que puede hacer todas estas cosas. Para proteger sus derechos necesitamos algunas restricciones que prohiban a cualquiera negarle a usted estos derechos o pedirle que renuncie a ellos. Estas restricciones se traducen en ciertas obligaciones que le afectan si distribuye copias del software, o si lo modifica. Por ejemplo, si distribuye copias de uno de estos programas, sea gratuitamente, o a cambio de una contraprestación, debe dar a los receptores todos los derechos que tiene. Debe asegurarse de que ellos también reciben, o pueden conseguir, el código fuente. Y debe mostrarles estas condiciones de forma que conozcan sus derechos. Protegemos sus derechos con la combinación de dos medidas: Ponemos el software bajo copyright y le ofrecemos esta licencia, que le da permiso legal para copiar, distribuir y/o modificar el software. También, para la protección de cada autor y la nuestra propia, queremos asegurarnos de que todo el mundo comprende que no se proporciona ninguna garantía para este software libre. Si el software se modifica por cualquiera y éste a su vez lo distribuye, queremos que sus receptores sepan que lo que tienen no es el original, de forma que cualquier problema introducido por otros no afecte a la reputación de los autores originales. Por último, cualquier programa libre está constantemente amenazado por patentes sobre el software. Queremos evitar el peligro de que los redistribuidores de un programa libre obtengan patentes por su cuenta, convirtiendo de facto el programa en propietario. Para evitar esto, hemos dejado claro que cualquier patente debe ser pedida para el uso libre de cualquiera, o no ser pedida. Los términos exactos y las condiciones para la copia, distribución y modificación se exponen a continuación.

Términos y condiciones para la copia, distribución y modificación

Esta Licencia se aplica a cualquier programa u otro tipo de trabajo que contenga una nota colocada por el tenedor del copyright diciendo que puede ser distribuido bajo los términos de esta Licencia Pública General. En adelante, «Programa» se referirá a cualquier programa o trabajo que cumpla esa condición y «trabajo basado en el Programa» se referirá bien al Programa o a cualquier trabajo derivado de él según la ley de copyright. Esto es, un trabajo que contenga el programa o una proción de él, bien en forma literal o con modificaciones y/o traducido en otro lenguaje. Por lo tanto, la traducción está incluida sin limitaciones en el término «modificación». Cada concesionario (licenciatario) será denominado «usted». Cualquier otra actividad que no sea la copia, distribución o modificación no está cubierta por esta Licencia, está fuera de su ámbito. El acto de ejecutar el Programa no está restringido, y los resultados del Programa están cubiertos únicamente si sus contenidos constituyen un trabajo basado en el Programa, independientemente de haberlo producido mediante la ejecución del programa. El que esto se cumpla, depende de lo que haga el programa. Usted puede copiar y distribuir copias literales del código fuente del Programa, según lo has recibido, en cualquier medio, supuesto que de forma adecuada y bien visible publique en cada copia un anuncio de copyright adecuado y un repudio de garantía, mantenga intactos todos los anuncios que se refieran a esta Licencia y a la ausencia de garantía, y proporcione a cualquier otro receptor del programa una copia de esta Licencia junto con el Programa. Puede cobrar un precio por el acto físico de transferir una copia, y puede, según su libre albedrío, ofrecer garantía a cambio de unos honorarios. Puede modificar su copia o copias del Programa o de cualquier porción de él, formando de esta manera un trabajo basado en el Programa, y copiar y distribuir esa modificación o trabajo bajo los términos del apartado 1, antedicho, supuesto que además cumpla las siguientes condiciones: Debe hacer que los ficheros modificados lleven anuncios prominentes indicando que los ha cambiado y la fecha de cualquier cambio. Debe hacer que cualquier trabajo que distribuya o publique y que en todo o en parte contenga o sea derivado del Programa o de cualquier parte de él sea licenciada como un todo, sin carga alguna, a todas las terceras partes y bajo los términos de esta Licencia. Si el programa modificado lee normalmente órdenes interactivamente cuando es ejecutado, debe hacer que, cuando comience su ejecución para ese uso interactivo de la forma más habitual, muestre o escriba un mensaje que incluya un anuncio de copyright y un anuncio de que no se ofrece ninguna garantía (o por el contrario que sí se ofrece garantía) y que los usuarios pueden redistribuir el programa bajo estas condiciones, e indicando al usuario cómo ver una copia de esta licencia. (Excepción: si el propio programa es interactivo pero normalmente no muestra ese anuncio, no se requiere que su trabajo basado en el Programa muestre ningún anuncio). Estos requisitos se aplican al trabajo modificado como un todo. Si partes identificables de ese trabajo no son derivadas del Programa, y pueden, razonablemente, ser consideradas trabajos independientes y separados por ellos mismos, entonces esta Licencia y sus términos no se aplican a esas partes cuando sean distribuidas como trabajos separados. Pero cuando distribuya esas mismas secciones como partes de un todo que es un trabajo basado en el Programa, la distribución del todo debe ser según los términos de esta licencia, cuyos permisos para otros licenciatarios se extienden al todo completo, y por lo tanto a todas y cada una de sus partes, con independencia de quién la escribió.

Por lo tanto, no es la intención de este apartado reclamar derechos o desafiar sus derechos sobre trabajos escritos totalmente por usted mismo. El intento es ejercer el derecho a controlar la distribución de trabajos derivados o colectivos basados en el Programa. Además, el simple hecho de reunir un trabajo no basado en el Programa con el Programa (o con un trabajo basado en el Programa) en un volumen de almacenamiento o en un medio de distribución no hace que dicho trabajo entre dentro del ámbito cubierto por esta Licencia. Puede copiar y distribuir el Programa (o un trabajo basado en él, según se especifica en el apartado 2, como código objeto o en formato ejecutable según los términos de los apartados 1 y 2, supuesto que además cumpla una de las siguientes condiciones: Acompañarlo con el código fuente completo correspondiente, en formato electrónico, que debe ser distribuido según se especifica en los apartados 1 y 2 de esta Licencia en un medio habitualmente utilizado para el intercambio de programas, o Acompañarlo con una oferta por escrito, válida durante al menos tres años, de proporcionar a cualquier tercera parte una copia completa en formato electrónico del código fuente correspondiente, a un coste no mayor que el de realizar físicamente la distribución del fuente, que será distribuido bajo las condiciones descritas en los apartados 1 y 2 anteriores, en un medio habitualmente utilizado para el intercambio de programas, o Acompañarlo con la información que recibiste ofreciendo distribuir el código fuente correspondiente. (Esta opción se permite sólo para distribución no comercial y sólo si usted recibió el programa como código objeto o en formato ejecutable con tal oferta, de acuerdo con el apartado b anterior). Por código fuente de un trabajo se entiende la forma preferida del trabajo cuando se le hacen modificaciones. Para un trabajo ejecutable, se entiende por código fuente completo todo el código fuente para todos los módulos que contiene, más cualquier fichero asociado de definición de interfaces, más los guiones utilizados para controlar la compilación e instalación del ejecutable. Como excepción especial el código fuente distribuido no necesita incluir nada que sea distribuido normalmente (bien como fuente, bien en forma binaria) con los componentes principales (compilador, kernel y similares) del sistema operativo en el cual funciona el ejecutable, a no ser que el propio componente acompañe al ejecutable. Si la distribución del ejecutable o del código objeto se hace mediante la oferta acceso para copiarlo de un cierto lugar, entonces se considera la oferta de acceso para copiar el código fuente del mismo lugar como distribución del código fuente, incluso aunque terceras partes no estén forzadas a copiar el fuente junto con el código objeto. No puede copiar, modificar, sublicenciar o distribuir el Programa excepto como prevé expresamente esta Licencia. Cualquier intento de copiar, modificar sublicenciar o distribuir el Programa de otra forma es inválida, y hará que cesen automáticamente los derechos que te proporciona esta Licencia. En cualquier caso, las partes que hayan recibido copias o derechos de usted bajo esta Licencia no cesarán en sus derechos mientras esas partes continúen cumpliéndola. No está obligado a aceptar esta licencia, ya que no la ha firmado. Sin embargo, no hay hada más que le proporcione permiso para modificar o distribuir el Programa o sus trabajos derivados. Estas acciones están prohibidas por la ley si no acepta esta Licencia. Por lo tanto, si modifica o distribuye el Programa (o cualquier trabajo basado en el Programa), está indicando que acepta esta Licencia para poder hacerlo, y todos sus términos y condiciones para copiar, distribuir o modificar el Programa o trabajos basados en él. Cada vez que redistribuya el Programa (o cualquier trabajo basado en el Programa), el receptor recibe automáticamente una licencia del licenciatario original para copiar, distribuir o modificar el Programa, de forma sujeta a estos términos y condiciones. No puede imponer al receptor ninguna restricción más sobre el ejercicio de los derechos aquí garantizados. No es usted responsable de

hacer cumplir esta licencia por terceras partes. Si como consecuencia de una resolución judicial o de una alegación de infracción de patente o por cualquier otra razón (no limitada a asuntos relacionados con patentes) se le imponen condiciones (ya sea por mandato judicial, por acuerdo o por cualquier otra causa) que contradigan las condiciones de esta Licencia, ello no le exime de cumplir las condiciones de esta Licencia. Si no puede realizar distribuciones de forma que se satisfagan simultáneamente sus obligaciones bajo esta licencia y cualquier otra obligación pertinente entonces, como consecuencia, no puede distribuir el Programa de ninguna forma. Por ejemplo, si una patente no permite la redistribución libre de derechos de autor del Programa por parte de todos aquellos que reciban copias directa o indirectamente a través de usted, entonces la única forma en que podría satisfacer tanto esa condición como esta Licencia sería evitar completamente la distribución del Programa. Si cualquier porción de este apartado se considera inválida o imposible de cumplir bajo cualquier circunstancia particular ha de cumplirse el resto y la sección por entero ha de cumplirse en cualquier otra circunstancia. No es el propósito de este apartado inducirle a infringir ninguna reivindicación de patente ni de ningún otro derecho de propiedad o impugnar la validez de ninguna de dichas reivindicaciones. Este apartado tiene el único propósito de proteger la integridad del sistema de distribución de software libre, que se realiza mediante prácticas de licencia pública. Mucha gente ha hecho contribuciones generosas a la gran variedad de software distribuido mediante ese sistema con la confianza de que el sistema se aplicará consistentemente. Será el autor/donante quien decida si quiere distribuir software mediante cualquier otro sistema y una licencia no puede imponer esa elección. Este apartado pretende dejar completamente claro lo que se cree que es una consecuencia del resto de esta Licencia. Si la distribución y/o uso de el Programa está restringida en ciertos países, bien por patentes o por interfaces bajo copyright, el tenedor del copyright que coloca este Programa bajo esta Licencia puede añadir una limitación explícita de distribución geográfica excluyendo esos países, de forma que la distribución se permita sólo en o entre los países no excluidos de esta manera. En ese caso, esta Licencia incorporará la limitación como si estuviese escrita en el cuerpo de esta Licencia. La Free Software Foundation puede publicar versiones revisadas y/o nuevas de la Licencia Pública General de tiempo en tiempo. Dichas nuevas versiones serán similares en espíritu a la presente versión, pero pueden ser diferentes en detalles para considerar nuevos problemas o situaciones. Cada versión recibe un número de versión que la distingue de otras. Si el Programa especifica un número de versión de esta Licencia que se refiere a ella y a «cualquier versión posterior», tienes la opción de seguir los términos y condiciones, bien de esa versión, bien de cualquier versión posterior publicada por la Free Software Foundation. Si el Programa no especifica un número de versión de esta Licencia, puedes escoger cualquier versión publicada por la Free Software Foundation. Si quiere incorporar partes del Programa en otros programas libres cuyas condiciones de distribución son diferentes, escribe al autor para pedirle permiso. Si el software tiene copyright de la Free Software Foundation, escribe a la Free Software Foundation: algunas veces hacemos excepciones en estos casos. Nuestra decisión estará guiada por el doble objetivo de de preservar la libertad de todos los derivados de nuestro software libre y promover el que se comparta y reutilice el software en general.

Ausencia de Garantía

Como el programa se licencia libre de cargas, no se ofrece ninguna garantía sobre el programa, en todas la extensión permitida por la legislación aplicable. Excepto cuando se indique de otra forma por escrito, los tenedores del copyright y/u otras partes proporcionan el programa «tal cual», sin garantía de ninguna clase, bien expresa o implícita, con inclusión, pero sin limitación a las garantías mercantiles implícitas o a la conveniencia para un propósito particular. Cualquier riesgo referente a la calidad y prestaciones del programa es asumido por usted. Si se probase que el Programa es defectuoso, asume el coste de cualquier servicio, reparación o corrección. En ningún caso, salvo que lo requiera la legislación aplicable o haya sido acordado por escrito, ningún tenedor del copyright ni ninguna otra parte que modifique y/o redistribuya el Programa según se permite en esta Licencia será responsable ante usted por daños, incluyendo cualquier daño general, especial, incidental o resultante producido por el uso o la imposibilidad de uso del Programa (con inclusión, pero sin limitación a la pérdida de datos o a la generación incorrecta de datos o a pérdidas sufridas por usted o por terceras partes o a un fallo del Programa al funcionar en combinación con cualquier otro programa), incluso si dicho tenedor u otra parte ha sido advertido de la posibilidad de dichos daños.

Quiénes Somos

Proyecto Sodero está formado por:

- Micaela Antelo
- Matias Blasi
- Hernán Mazzoni
- Sebastián Zaffarano

Somos un grupo de estudiantes de la **Universidad Nacional de la Matanza** que estamos cursando el último año de la carrera **Ingeniería en Informática** y encaramos este proyecto, entre otras razones para la aprobación de la **última** materia de nuestra carrera, denominada justamente "Proyecto", a cargo del Lic. Tomassino.

Manual del Usuario

Índice

Proyecto SODERO

Es claramente conocida la importancia de la investigación en nuestra profesión, es por ello que consideramos importante el desarrollar un proyecto que permita ampliar los conocimientos tanto a nivel de programación como el conocimiento del funcionamiento de un sistema operativo. Es por eso que pensamos en desarrollar un Sistema Operativo GNU. En el cuál puedan participar todos aquellos profesionales y/o estudiantes que lo deseen, con el único fin programar y comprender la arquitectura intel, la programación en assembler y el funcionamiento de un sistema operativo con mayor nivel de detalle.

Un punto importante es que SODERO será gratuito. De hecho, el sistema básico se podrá conseguir en Internet sin costo. Tampoco exigirá la compra de licencias para su uso, por lo que será absolutamente libre, dado que se regirá por la Licencia Pública General (GPL en inglés). Lo anterior significa que puede ser instalado en muchos computadores a partir de una sola distribución, sin tener que pagar adicionales por cada implementación. La Licencia Pública General es el icono mundial de lo que se denomina Open Source, lo que podríamos entender como software de fuente libre o abierta. El concepto es muy sencillo. Los programas deben acompañarse con el código fuente que los hace funcionar. Esto permite que los usuarios puedan hacer cambios, corrijan errores y mejoren las versiones originales. Como resultado de ello, el software se desarrolla rápida y eficazmente. El software libre utiliza los conocimientos de programadores del mundo entero, en vez de unos pocos que trabajan encerrados en una empresa. Las correcciones aparecen de la noche a la mañana, en vez de años después. Lo único que se pide es que dichos resultados se publiquen para que otros puedan disfrutarlos.

En cuanto a las características del sistema podemos decir que será un sistema operativo con un kernel monolítico, un conjunto de llamas al sistema y un shell para su operación. Trabajará en modo protegido. Será multitarea (es decir que puede haber más de una tarea corriendo al mismo tiempo) y monousuario (solo puede haber una persona utilizando el sistema al mismo tiempo).

Si deseas participar de este proyecto visítanos en www.proyecto-sodero.com.ar, en dónde podrás encontrar entre otras cosas los fundamentos teóricos, la documentación recopilada, el fuente desarrollado, los avances y dificultades encontradas, el manual del programador, etc.
Contamos con tu aporte!!.

Manual del Usuario

Organización

Dividimos las etapas, esencialmente, en tres: investigación y capacitación, desarrollo y paralelamente a ambas: documentación. En la web del proyecto se puede encontrar todo lo documentado sobre el mismo. (www.proyecto-sodero.com.ar) La etapa de capacitación comenzó el 12 de Febrero de 2002 La subdividimos en los siguientes items:

- Arquitectura de microprocesadores
- Programación assembler
- Programación assembler en modo protegido
- Teoría de sistemas operativos

Para la etapa de desarrollo, cada uno se fijo una cantidad de horas semanales para realizar pr&ácticas puntuales de assembler, y luego con reuniones convenidas, comentamos y juntamos lo realizado. La documentación del proyecto consiste básicamente en dos manuales: sodero_doc.txt y sodero_tecnico.txt, complementándose con un documento "pendientes.txt" el cual detalla el avance punto a punto del proyecto y las próximas mejoras a realizar. Los mismos se fueron confeccionando a la par del avance en el proyecto.

Manual del Usuario

Seguimiento del proyecto

La etapa de desarrollo comenzo el sábado 01 de Junio de 2002:
El primer paso fue realizar un bootstrap que simplemente realice un "Hola Mundo". Luego le agregamos al mismo el pasaje a modo protegido. Posteriormente modularizamos el bootstrap, loader y kernel.

Manual Técnico

Índice

Manual Técnico

Convenciones

1. Documentación
    1. Párrafos
        a. Los párrafos tendrán una sangría de 4 espacios.
        b. No se deben superar las 80 columnas.
        c. Solo se podrán utilizar carácteres de código ASCII superior al >= 32 y <= 126
    2. Títulos y enumeraciones
        a. Todo párrafo, título, subtitulo o enumeración Comenzará con mayúscula.
        b. En cada título o subtitulo se indicará con un número correlativo su orden o precedencia, y las enumeraciones con una letra del abecedario:

```
1. Título 1
   1.1. Subtitulo 1
      a. enumeración 1
      b. enumeración 2
   1.2. Subtitulo 2
      a. enumeración 1
      b. enumeración 2
```

2. Programación en Assembler
    1. Nombres de variables, métodos, subrutinas, etc.
    Un fragmento de código en lenguaje ensamblador esta formado por mnemónicos, etiquetas y operandos. Asimismo, las etiquetas pueden ser locales a una subrutina, o globales a todo el módulo.
    2. Definición lexicógrafica de los nombres
    Definimos que los nombres de variables siempre irán en minúsculas, y en el caso que un nombre este compuesto por mas de una palabra, las mismas se separaran con un guión bajo, por ejemplo:

```
contador         dw      0x00
```

    Los nombres de las constantes irán en mayúsculas, separando las palabras con un guión bajo:

```
SELECTOR_SEGMENTO        equ     0x123
```

    Para declarar subrutinas (asembler) o funciones (C), se respetará lo dicho para los nombres de variables:

```
lee_disco:
```

    En el caso de las etiquetas, tenemos tres tipos:
        I. Etiquetas globales: las mismas se declaran en forma análoga a las variables
        II. Etiquetas locales a una subrutina: también se declaran igual que una variable, pero con la salvedad que deben comenzar con un punto ('.'), permitiendo de esa forma repetir nombres en diferentes subrutinas.
        III. Etiquetas definidas dentro de una macro: deben comenzar con un doble signo porciento ('%%') y también cumplir con los mismos requerimientos de las variables.
    3. Definición sintáctica de los nombres
    Para las funciones o subrutinas, habrá que declarar nombres que contengan algún verbo definiendo la acción de la misma, por ejemplo:

```
mostrar_palabra:
```

    Cuando declaremos variables o constantes, habrá que nombrarlas con sustantivos,

que indiquen lo que representan las mismas:

```
posicion_inicial        dw      0x00
SEGMENTO_VIDEO          equ     0x0B800
```

En el caso de las etiquetas, NO utilizar nombres 'genéricos' (por ejemplo .l1), sino que los mismos deben dar un conocimiento sobre lo que representan:

```
        cmp     eax, valor
        jne     .fin_iteraccion
.fin_iteraccion
```

4. Indentación
   En lenguaje ensamblador, no tenemos anidamientos de bloques como en los lenguajes de alto nivel; podría decirse que únicamente hay dos niveles de indentación: etiquetas/subrutinas y mnemónicos

```
main:
        push    eax
        pop     ss
        cmp     eax, 10
        jz      .opcion_dos     ; comentarios
.opcion_uno
        xor     eax, eax
.opcion_dos
        mov     ebx, eax
```

   Las etiquetas y variables irán sin indentación alguna, pegadas al margen izquierdo. Separados un tabulador del marquen izquierdo colocaremos los mnemónicos y con un tabulador de distancia irán el/los operando/s, si los hubiese.
5. Comentarios
   Los comentarios irán separados 'N' tabs de los operandos, siendo 'N' lo necesario para que todos los comentarios de un bloque queden alineados. Tenemos tres tipos de comentarios:
   I. Encabezados de funciones/macros: deben cumplir con el siguiente formato

```
;/*****************************************************************
; * nombre_funcion: "descripcion"
; *
; * entrada:
; *
; * salida:
; *
; * registros que destruye/modifica
; *
; * versión:
; * fecha de creación:
; * ultima fecha de modificación:
; *****************************************************************
```

   Es importante aclarar que la descripción de la función debe decir QUE hace la misma y no COMO lo hace. Tanto entrada como salida serán una lista de registros con su significado dentro de la subrutina, o eventualmente la posición dentro del stack (para pasaje de parámetros a través de la pila)

   II. Comentarios en un bloque: En el caso que haya un fragmento de código cuya lógica se considere "complicada" o confusa, al comienzo del mismo se relizará un comentario de cual es la funcionalidad del mismo:

```
;
```

```
; realiza la conversión entre el formato CSH y LBA de un cluster
;
.pasaje_a_lba:
        mov     eax, ecx
        push    edx
        pop     ebx
```

III. Comentarios en una línea: Salvo las líneas triviales, intentar comentar cada una de las líneas de código, con el objeto de que cualquier persona que lo vea entienda que se quiere hacer.

```
        mov     ebx, [dividendo]        ; EBX = contenido de la variable
                                ; dividendo
        shr     ebx, 0x04               ; divido la variable dividendo por
                                ; cuatro.
```

6. Nombres de archivos:
   Los archivos deberán agrupar variables y subrutinas con funcionalidad común entre si, debiendo tener un nombre acorde a dicha funcionalidad.
   Si bien no tenemos restricciones con relación a la longitud del mismo, es preferible que los nombres no sean demasiado largos y de ser posible, que no este compuesto por mas de una palabra. En el caso de que haya que poner un nombre con dos o más palabras, intentar buscar la abreviatura de ambas y unirlas sin ningún carácter intermedio:

   ```
   kernsys.asm --> Kernel del Sistema
   ```

   Las extensiones serán:

   ```
   - .asm para todos los fuentes
   - .inc para los archivos con definición de macros y/o %define
   ```

3. Programación en C
   1. Nombres de variables, métodos, subrutinas, etc.
      En 'C' también tenemos nombres de variables, funciones, constantes, etc.
   2. Definición lexicográfica de los nombres
      En forma análoga a asembler, tenemos a las variables en minúsculas, y con un guión bajo separando palabras:

      ```
      int cantidad_alumnos = 100;
      double monto_del_aviso = 1;
      ```

      también las funciones y constantes cumplirán con las mismas restricciones:

      ```
      #define BUFFER 256
      #define MAX_INDICE 10

      int imprime_caracter (char c);
      ```

   3. Definición sintáctica de los nombres
      Remitirse a la sección análoga para lenguaje ensamblador
   4. Indentación
      A continuación se expone un fragmento de código en donde se pueden ver todas las restricciones de este punto:

      ```
      int main (int argc, char* argv) {
         int i, j = 0;
         for ( i = 0; i < 10; i++ ) {
            while (j++ < 20) {
               printf ("hola");
            }
      ```

```
    }
}
```

5. Comentarios
   Al comienzo de cada función debe ir un encabezado:

   ```
   /**
    * nombre_funcion: "descripción"
    *
    * versión:
    * fecha de creación:
    * última fecha de modificación: come
    */
   ```

   Es importante comenzar con '/**' y no con '/*' para utilizar la doc++ y de esa forma generar documentación adicional.
   Para un detalle un poco más exhaustivo remitirse a la sección 3 de la primera parte.
6. Nombres de archivos
   Al estar trabajando con ANSI C utilizaremos la extensión .c para referirnos a las fuentes y .h para las bibliotecas. Estas ultimas deberán tenés como primer y ultima línea respectivamente a;

   ```
   #ifndef _NOMBRE_H
   #define _NOMBRE_H

   #endif _NOMBRE_H // fin _NOMBRE_H
   ```

   El resto del criterio para archivos es exactamente el mismo que para lenguaje ensamblador

4. Utilizando el pendientes.txt
   Propongo la utilización de acá en más de un pendientes.txt, que sea totalmente acumulativo con el siguiente formato:

   ```
   //**************************
   pendientes:

          1 - descripción 1
          2 - descripción 2

   //**************************
   ```

5. Utilizando el log.txt
   Del mismo modo que se debe llevar actualizado el archivo 'pendientes.txt' debemos llevar actualizado el archivo 'log.txt' de la siguiente manera:

   ```
   //**************************
   avances:

         fecha2 hora2    descripción de la mejora2
         fecha1 hora1    descripción de la mejora1

   Próximas mejoras:


   Limitaciones:


   //**************************
   ```

Manual Técnico

Especificaciones Técnicas

1. Sistema de archivos

   Inicialmente hubo una propuesta de realizar un sistema de archivos orientado a objetos, como ejemplo de la intención de realizar algo totalmente "nuestro" y diferente.
   Con el paso de los días, y resignando algunas ideas para mas adelante decidimos, para avanzar de a poco, comenzar soportando el sistema de archivos FAT. Siempre teniendo en cuenta de darle al sistema operativo la capacidad de adaptarse a cualquier nuevo sistema de archivos. Esta propuesta se puede observar en el documento saoo.txt

2. Formato de binarios

3. Administración de procesos

4. Manejo de memoria

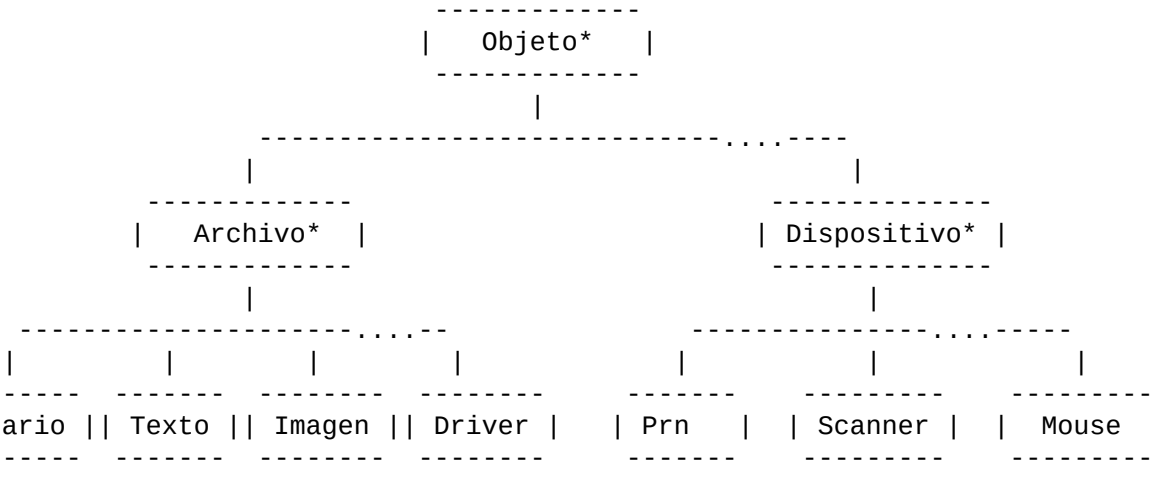
5. Interrupciones y excepciones

Y si hacemos un SISTEMA OPERATIVO???

Algunas ideas para el sistema operativo: (sin nombre todavia)

22:40 27/03/2002 A esta altura, con cero conocimiento de assembler, algo mas de sistemas y de programacion se me ocurren algunas caracteristicas que me gustaria que tenga el SO:

Primero una orientacion a objetos en TODO aquello que sea aplicable, hasta ahora no encontre item en el cual no se pueda aplicar:
Establecer para esto un diagrama de gerarquia de clases:



* (Abstracta)

Todo este diagrama se me ocurre que puede representarse en tablas, al iniciar el SO, en el segmento de memoria del Kernel, alojar algunas tablas que me permitan saber la gerarquia de clases (HardCoded, para el punto de vista del usuario del SO, pero obviamente que en el desarrollo del Kernel, debera estar 100% parametrizado de la forma mas conveniente), para establecer los metodos y propiedades heredadas y el polimorfismo.

id	clase	Abstracta	Madre	Metodos	Propiedades	
1	Objeto	X	-	?	dueño	
2	Archivo	X	1	borrar	tamaño	
				copiar	ubicacion	
				mover		
3	Binario	-	2	ejecutar	?	
4	Texto	-	2	editar	?	
				imprimir		
5	Imegen	-	2	visualizar	resolucion	
				imprimir		
6	Driver	-	2	levantar	?	
				bajar		
7	Disposit.	X	1	levantar	driver	(deberia ser un ptro. a archivo de tipo driver)
8	Prn	-	7	imprimir	resolucion	
9	Scanner	-	7	scanear	resolucion	
10	Mouse	-	7	?	sencibiliada	

Lo de arriba es solo una idea llena de ejemplos. La idea que trato de explicar es la de implementar herencia, y encapsulamiento en todo el entorno del SO. Se que suena muy complicado, y probablemente lo sea, pero a priori, me parece sencillo visto de esta forma: (espero opiniones, puede que este diciendo barbaridades) ;)

(tener en cuenta que para el caso de los metodos lo que se deberia almacenar es lo que conocemos como un puntero a funcion, supongo que seria algo asi como la direccion de memoria donde existe el codigo correspondiente al procedimiento)

Para el caso de los objetos tipo archivos, implicaria guardar en el disco, para cada tipo de archivo una cabecera diferente en funcion del tipo de archivo que se trate, esa estructura indicaria el tipo, y tendria las propiedades correspondientes al tipo y las heredadas.
Y luego simplemente cuando el usuario escriba: NombreArchivo.ejecutar, habria que verificar el tipo de archivo de 'NombreArchivo', y luego si ese tipo contiene el metodo ejecutar en su clase o superclases. En tal caso se llamaria a la rutina de ejecutar con el puntero (ubicacion) de 'NombreArchivo' como parametro (seguramente almacenandolo en algun registro).
Algo muy parecido se realizaria con las propiedades, solo que luego de verificar el tipo de archivo y corroborar que existe la propiedad deseada en ese tipo o supertipo, ya conoceriamos su estructura, (por lo menos su cabecera), en la cual se encuentra el valor de la propiedad con algun offset a partir de la posicion inicial del archivo en el disco y habria que modificar dicha propiedad escribiendo en el disco.

De mas esta decir que al crear un objeto se debe indicar el tipo de algun modo y habra que verificar que no sea abstracto.
Al momento de crear archivos se requiria el tipo, con lo cual sabriamos crear la cabecera en el disco, por ejemplo si quiero crear una imagen:

siguiendo la tabla: id 5; dueño ilcapo; tamaño 10; ubicacion xxx; resolucion xxx;

Si el archivo se llama imagen1:

imagen1.ejetucar daria error por ser de tipo 5 y no tener el metodo ejecutar
imagen1.resolucion = 6 modificaria la cabecera del archivo en el disco, previa verificacion de tipo de dato y existencia de propiedad resolucion en el mismo
imagen1.visualizar verifico que para el tipo 5 existe el metodo visualizar, por lo tanto almaceno en algun registro el puntero de dicho archivo y ejecuto la rutina de visualizar, la cual hara lo que deba hacer...
imagen1.borrar lo borro
imagen1.copiar imagen2.ubicacion copiar es metodo de archivo, supertipo de imagen, esto haria una copia de imagen1 en la ubicacion de la imagen2
imagen1.dueno = usr1 dueno es propiedad de la clase objeto, supertipo de archivo, supertipo de imagen, por lo tanto correcto, actualizo cabecera de imagen1 en disco

Con algun criterio similar, que aun no pense demasiado habria que tratar a los dispositivos, por ejemplo:

prn1.levantar configuraria, prenderia la impresora, pero para esto habria que conocer mucho de cada dispositivo y crear los drivers correspondientes, pero este metodo deberia hacer lo siguiente:

```
|
\----- (prn1.driver).levantar     la cazaron???
```

prn1.imprimir imagen1 seguramente equivalente a imagen1.imprimir

y asi.....

Solo tengo una gran duda: ¿No es esto encapsulamiento, herencia y polimorfismo en su

totalidad? ¿O le estoy pifiando?

Seguimiento del Proyecto

seb 12:30 26/09/2002 Implementación del handler de interrión de impresión.
mat 13:45 24/09/2002 Creación de la IDT de modo protegido.
her 16:44 21/08/2002 Creación de registros de tareas representativos del kernel y de una tarea usuario (TSS's)
seb 20:10 11/08/2002 Integracion loader (asm) + kernel ("C")
mat 21:50 20/07/2002 Intento fallido de reutilización de las interrupciones del bios.
seb 14:00 11/07/2002 Actualización de las opciones del site: "Convenciones", "Especificaciones tecnicas", "Organizacion", "Seguimiento" y "Pendientes"
seb 16:30 07/07/2002 Cambio en el look&feel del site Actualizacion del contenido de las opciones "Introduccion", "Quienes Somos" y "Objetivos"
seb 22:20 27/06/2002 Agregar en la estructura de directorios proyecto_sodero/doc el directorio correspondiente al sitio web, asi cualquiera que hace algo relacionado con la documentacion lo programa directamente sobre la pagina
seb 10:30 25/06/2002 Modificacion del bootstrap indicando en %defines las posiciones donde cargar el loader+kernel
seb 14:00 21/06/2002 Identificacion de un BUG en el loader/kernel!!!!! ante determinadas circunstancias se comporta extranio, booteando la PC, supongo que se debe a una MALA definicion de los segmentos y la GDT
seb 12:00 21/06/2002 Redefinicion de la tabla de interrupciones para habilitar la int 0x09 para impresion (esto al final se volvio atras debido a que aun no podemos deshacernos de todas las interrupciones, eventualmente usaremos una subrutina imprimir que haga lo mismo)
seb 10:00 21/06/2002 Pasaje a modo protegido dentro del kernel/loader
seb 12:00 20/06/2002 Realizacion de un "template" para la pagina web del site, dejando parametrizados los colores y fuentes, permitiendo de esa forma modificar facilmente el look&feel de la pagina.
seb 15:00 19/06/2002 Cargar en memoria el kernel/loader y hacer un jmp hacia la primer posicion de memoria del mismo para que comience a ejecutar.
seb 14:00 19/06/2002 Interpretacion de archivos FAT12 desde assembler: leer y cargar a memoria un archivo pasado como parametro
mat 20:39 22/06/2002 Modificacion en macro 'imprimir' para que resguarde el valor de bx
mat 18:00 20/06/2002 Corregi bug en macro 'obtener_limite'
mat 14:50 20/06/2002 Creacion de una macro para modificar una entrada de la idt 'modificar_idt'
mat 14:45 20/06/2002 creacion de una macro para reservar espacio para descriptores de interrupciones 'nuevo_descriptor_int'
mat 18:00 18/06/2002 Creacion de una macro que imprime en pantalla el valor de cada uno de los registros de la CPU, 'ver_regs'
mat 12:00 17/06/2002 Cearcion de una macro para transformar el numero binario existente en eax a un numero de otra base cualquiera (>= 2 y <= 36), ubicado en memoria y referenciado por si,

'binario_a_valor'
mat 10:00 14/06/2002 Cearcion de una macro para transformar el numero binario existente en eax al correspondiente numero en hexa formato BCD, ubicado en memoria y referenciado por si, 'binario_a_hexa'
mat 13:00 07/06/2002 Creacion de una macro para obtener el limite de un descriptor de segmento y sumarselo a eax, 'obtener_limite'
mat 09:00 06/06/2002 Creacion de una macro para obtener la base de un descriptor de segmento y guardarla en eax, 'obtener_base'
seb 20:30 04/06/2002 creacion de una subrutina para impresion y manejo de la misma a traves de la interrupcion 0x9
mat 12:58 02/06/2002 pasaje a modo protegido desde el bootstrap
seb 12:58 01/06/2002 realizacion del bootstrap con un simple "Hola Mundo"
seb 09:12 23/05/2002 establecimos el standard de programacion en assembler
seb 09:55 27/04/2002 abrimos una web para publicar los avances de nuestro proyecto 23:45 27/03/2002 asignamos un nombre al proyecto: SODERO

Pendientes

- ver por que no pueden concurrir dos interfaces
- coordinar las lucesitas del teclado al inicio, y dejar numlock encendido
- definir y desarrollar system calls

    * abrir_archivo
    * leer_archivo
    * cerrar_archivo
    * liberar
    * cambiar_directorio
    * beep
    * escribir_archivo

- procesos

    * cat
    * cambiar_directorio
    * cdplay
    * sys
    * borrar_pantalla

- Uso de Fork
- driver de disquetera (cuasi andando!!! mejorar leer_archivo y manejo errores)
- bootstrap -> loader -> kernel
- analizar posibilidad de copiar el loader a 0x90000 desde el bootstrap en lugar de copiarlo a 0x7E0 y luego moverlo
- lograr 8188 tareas.
- ver las alternativas posibles al calculo de la base de alocacion de memoria
- imprimir con argumentos variables (arreglar bug)
- implementar paginacion
- algoritmos:

alocacion de memoria
asignacion en la GDT
scheduling

Código Fuente e Instalador del Proyecto

sodero.fuentes.0.01.tgz
sodero.bin.0.01.tgz

Enlaces

Sitios de Interés para visitar:

- http://nondot.org/sabre/os/articles/
- http://www.thesky.demon.co.uk/os-development/index.html
- http://www.execpc.com/~geezer/os/
- http://www.faqs.org/faqs/assembly-language/x86/general/part1/section-13.html
- http://www.execpc.com/~geezer/os/pm.htm
- http://x86.ddj.com/articles/pmbasics/tspec_a1_doc.htm
- http://phantom.urbis.net.il/bphantom/pmode-l_FAQ.html
- ftp://x2ftp.oulu.fi/pub/msdos/programming/pmode/00index.html
- http://nondot.org/sabre/os/files/ProtectedMode/segments.html
- http://linuxassembly.org/self.html
- http://www.menuetos.org
- http://www.gnu.org