Creación de Hilos Virtuales para la Emulación de DOS en Mach en el ambiente orientado a objetos de NeXTSTEP

Amilcar Meneses Viveros **Hugo García Monroy**
Departamento de Aplicación de Microcomputadoras
Instituto de Ciencias
Universidad Autónoma de Puebla.
Apdo. Postal 461, C.P.72000, Puebla, Puebla, México
aviveros@mail.cinvestav.mx gmonroy@servidor.unam.mx

Resumen

Cuando se está trabajando en un ambiente determinado de un Sistema Operativo (SO), algunas veces es deseable que se pueda suministrar un ambiente diferente en el cuál una aplicación pueda ser ejecutada. Esto se puede conseguir mediante la simulación de un ambiente virtual de ejecución para la aplicación.

Los métodos para crear plataformas de simulación para Sistemas Operativos y las nuevas tecnologías de micronúcleo para desarrollar dichos sistemas están muy ligados, pues en ambos casos se resuelve el problema de ejecutar un SO a nivel de usuario. Las técnicas de simulación proponen dos alternativas para la ejecución de un SO en otra plataforma[1]: simular el hardware o modificar su núcleo. Por otra parte, la construcción de algunos de los nuevos sistemas operativos se basa en la tecnología de micronúcleo, lo que permite diseñarlos en base a servidores.

En este artículo se presenta el soporte para la emulación de DOS en el ambiente de desarrollo orientado a objetos NeXTSTEP, este soporte se realiza a través de una tarea virtual, de una tarea monitor y un hilo virtual. Para su ejecución se creó un objeto VM.app de la clase NXapp en el ambiente NeXTSTEP, él cual se comunica con los diferentes procesos creados para la ejecución de una aplicación de DOS.

1 Introducción

Los métodos para crear plataformas de simulación para Sistemas Operativos y las nuevas tecnologías de micronúcleo para desarrollar estos sistemas están muy ligados. Por un lado las técnicas de simulación proponen dos alternativas para la ejecución de un SO en otra plataforma; la primera técnica es simular el hardware para engañar al sistema operativo, y la segunda es modificar su núcleo para que este se ejecute en la otra arquitectura. Por otro lado, la construcción de los nuevos sistemas operativos se basa en la tecnología de micronúcleo. Un micronúcleo suministra la funcionalidad mínima de un sistema operativo (manejo de memoria, manejo de procesos e intercomunicacion entre procesos). Debido a esta característica, los micronúcleo se utilizan como base para la construcción de ambientes de sistemas operativos más complejos los cuales son diseñados en base a servidores.

Al montar un sistema operativo propio de algún hardware, como programa de aplicación de una plataforma (la plataforma es otro sistema operativo), desarrollada con la tecnología de micronúcleo, se mezclan los puntos anteriormente mencionados: la simulación de un SO, y la implantación del SO a través de micronúcleo.

En particular, es el caso de montar el sistema operativo DOS en el ambiente NeXTSTEP, debido a que DOS es nativo a la arquitectura de la PC 8086, y NeXTSTEP está desarrollado sobre el micronúcleo de Mach.

Los modos de trabajo de los diferentes microprocesadores de INTEL posteriores al 80286, definen las instrucciones y las caracteristicas de la arquitectura que son accesibles para la ejecución de los programas. Uno de éstos, es el modo virtual, el cual permite la emulación del 8086. Lo cual hace factible emular DOS como una tarea en el modo virtual, sobre un Sistema Operativo que se ejecuta en modo protegido . Esto es, el procesador 80386, ejecutará a DOS comportándose como si fuera un 8086.

Mach es un micronúcleo que puede ejecutarse en los procesadores de INTEL, a partir del 80386, y debido a las características que presenta el diseño del sistema operativo DOS, resulta atractivo poder montar este sistema operativo basándonos en Mach, además de que DOS es un sistema altamente popular y, por ende, tiene una gran cantidad de código ejecutable que es deseable seguir utilizando en otros ambientes. Como ambos sistemas se ejecutan en el mismo procesador, entonces para montar a DOS como una tarea de Mach, basta con hacer una emulación del hardware en el que se ejecuta, ya que la mayoría de las instrucciones se ejecutan directamente. Como la construcción de un sistema operativo planteado en la arquitectura de micronúcleo se basa en la construcción de diversos servidores (de manejo de video, del sistema de archivos, de puertos, y manejo de interrupciones, entre otros), la parte más importante es pues, el que soporta la emulación del modo 8086. Note que para la ejecución de DOS en Mach se realiza una emulación

[1] Esta plataforma es otro SO

y no una simulación, ya que en una simulación, el SO da la apariencia de ejecutarse sobre una máquina que no existe, y en la emulación el procesador 80386 se comportará como si fuera un 8086. Sin embargo nos apoyamos en algunas técnicas de simulación para proporcionarle a DOS, recursos que no tendrá disponibles al ejecutarse sobre un SO que trabaja el procesador en modo protegido.

2 Simulación de Sistemas Operativos

Las razones para simular un sistema operativo son diversas: el estudio del comportamiento del sistema, la ejecución de programas en una plataforma diferente sin necesidad de una recompilación del código fuente. entre otras [2].

Las dos alternativas principales para emular un SO son:

- El ambiente simula toda la máquina ejecutando el sistema operativo sobre la máquina.
- El ambiente modifica a el sistema operativo para que se ejecute en una máquina.

como se aprecia en la figura 1

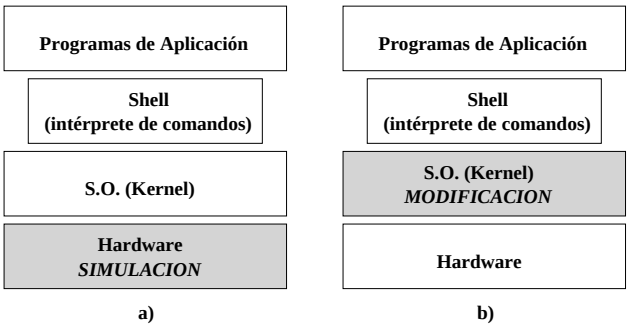

Figura 1: Tipos de Simulación para Sistemas Operativos.

La forma tradicional para ejecutar un sistema operativo a nivel usuario, es construir la simulación del hardware muy detallada. Esto incluye la simulación del CPU, de la unidad manejadora de memoria, y los dispositivos de entrada y salida accesados por el núcleo. El problema principal de esta técnica es que el Sistema simulado presenta un bajo rendimiento.

La alternativa para evitar la simulación del hardware es remover las partes del Sistema Operativo que no corre en nivel de usuario y reemplazarla por módulos que trabajen con la nueva arquitectura.

2.1 Simulación del hardware utilizando los servicios de un Sistema Operativo

Si consideramos a un Sistema Operativo como un programa con entradas y salidas, (ver figura 2), entonces las fuentes de entradas y salidas incluyen trap y excepciones que llegan desde el modo de usuario (como son llamados al sistema, fallas de página, errores aritméticos, etc.), interrupciones y DMA de los dispositivos. Las salidas de un sistema operativo controlan el MMU, el contenido de la memoria de usuario, y los dispositivos de entrada

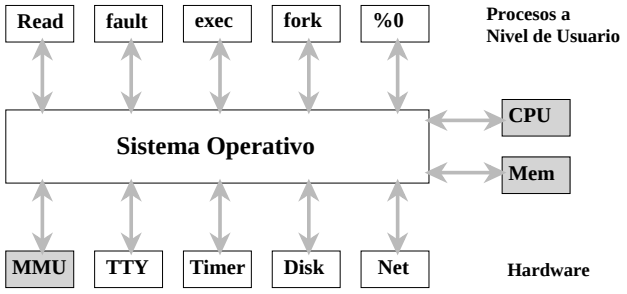

Figura 2: Interacciones de Entrada y Salida de un Sistema Operativo.

y salida. El CPU, MMU y la memoria tienen las ejecuciones más críticas. Si se ejecuta un Sistema Operativo en un ambiente de simulación, el ambiente proporciona las entradas y salidas esenciales, como lo haría una máquina real. El hardware se simula utilizando servicios proporcionados por un sistema operativo de propósito general, aunque lo que se obtiene con el hardware es diferente de lo que obtiene de un sistema operativo. Los Sistemas operativos modernos proporcionan un nivel de funcionalidad capaz de emular los servicios del hardware con una velocidad aceptable, sin embargo no proporcionan un medio para que los procesos de usuario obtengan el control de ciertos traps y excepciones. Similarmente, mientras los sistemas operativos modernos no proporcionan el acceso a la unidad manejadora de memoria (MMU) del hardware, proporcionan llamados para realizar mapeos a bloques de archivos.

Simulación del CPU. El servicio obvio que el sistema operativo utiliza para simular un CPU es la abstracción de proceso. Así, utilizando la abstracción de proceso para un CPU, es simple simular un multiprocesador utilizando un conjunto de procesos, uno por CPU. El número de CPU's simulados está limitado por número de procesos que soporte el sistema operativo.

Simulación de la arquitectura *trap* del CPU. La arquitectura trap del CPU permite que un Sistema Operativo tome el control de cualquier excepción o interrupción. Para procesos de nivel de usuario, el mecanismo de notificación de excepciones proporcionado por el sistema operativo es muy similar a una arquitectura *trap*. Algunos manejadores de *trap* se pueden simular a nivel usuario atrapando las señales y enviando la información al núcleo para que se simule. Una vez que se atrape una señal, se ejecutan las operaciones correspondientes y se reanuda la ejecución.

Manejo de las Instrucciones Privilegiadas. Las instrucciones privilegiadas y registros son otra característica utilizada por los sistemas operativos pero no son disponibles a nivel de usuario. Muchos CPU's soportan operaciones privilegiadas, como deshabilitar las interrupciones o cambiar el estado del MMU. Estas operaciones nunca se realizan a nivel de usuario. La simulación del

sistema operativo puede tratar con estas instrucciones en alguna de estas formas: reemplazar el código por instrucciones no privilegiadas para emular los efectos que la instrucción privilegiada ocasiona; o detectar la ejecución de la instrucción privilegiada y emular la instrucción en turno.

3 NeXSTEP y Mach

NeXTSTEP es un sistema operativo basado en UNIX, y está desarrollado sobre el micronúcleo de Mach.

3.1 Mach

Mach es un micronúcleo desarrollado en la Universidad de Carnegie Mellon y es compatible con la versión 4.3 de UNIX BSD. Mach se encarga de suministrar los recursos básicos de un sistema operativo, estos recursos son: manejo de procesos, manejo de memoria virtual y la comunicación entre procesos [7]. Los demás recursos del sistema operativo se ejecutan como procesos a nivel de usuario (como el sistema de archivo y además se pueden definir algunos manejadores de dispositivos). Mach se utiliza como plataforma para desarrollar sistemas operativos (figura 3).

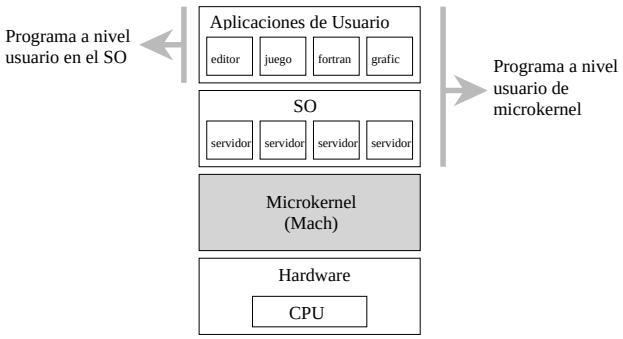

Figura 3: Implantación de un SO a partir de un micronúcleo.

Mach se apoya en cinco abstracciones para suministrar sus recursos básicos [8]:

Tarea Una tarea es una colección de recursos del sistema, estos recursos incluyen un espacio de direcciones virtuales e hilos entre otros. Se hace referencia a estos recursos, con excepción del espacio de direcciones, mediante puertos. Estos recursos se pueden compartir con otras tareas siempre que se distribuyan los derechos al puerto. El espacio de direcciones virtuales puede manejar hasta 4 GB de memoria, referenciado por direcciones de máquina. Partes de este espacio se pueden compartir, heredar o manejar como memoria externa. Una tarea puede contener varios hilos.

Hilo Un hilo es un punto de control del flujo en una tarea. Tiene accesos a todos los elementos que tenga la tarea. Se ejecuta potencialmente en paralelo siempre que otros hilos se encuentren en la misma tarea.

Puerto Un puerto es un canal de comunicación unidireccional entre un cliente que solicita un servicio y un servidor que proporciona dicho servicio. Los puertos y mensajes constituyen una de las formas de comunicación entre hilos y tareas.

Mensaje Un mensaje es una colección de datos con un formato que se pasa entre dos entidades a través de sus puertos. Un mensaje puede tener derechos de puertos (además de los datos) y de esta manera una tarea puede obtener nuevos derechos por el hecho de recibir un mensaje.

Objeto de Memoria Esta unidad sirve como un medio para mapear regiones de memoria a otra tarea.

Mach no proporciona la noción tradicional de proceso, en su lugar tiene dos nociones para soportar un proceso: las tareas y los hilos. La noción de un hilo de Mach es la de un punto de control. Una tarea existe para proporcionar los recursos para contener a los hilos. Esta división se hace para soportar el paralelismo y compartir recursos.

Por otro lado, la facilidad del manejo de excepciones de Mach permite que el usuario defina y maneje sus propias excepciones, agregándolas a las definidas por el sistema. El manejador de excepciones nunca se ejecuta en el contexto del hilo víctima, y el micronúcleo es el encargado de notificar al manejador cuando el hilo víctima a producido un excepción. La comunicación para el manejo de excepciones se realiza a través de mensajes entre el manejador de excepciones y el hilo víctima. Así las primitivas de *atrapar, esperar, notificar* y *limpiar* constituyen un llamado a un procedimiento remoto (RPC). El manejador de excepciones se implementa utilizando un RPC basado en mensajes. La excepción RPC consiste de dos mensajes: un mensaje inicial que invoca al RPC y un mensaje de respuesta que lo completa. El mensaje inicial contine los siguientes elementos:

- Puertos de envío y respuesta para el RPC.
- Las identidades de la tarea e hilo que ocasionan la excepción.
- Una clase excepción independiente de la máquina.
- Dos campos dependientes de la máquina que identifican la excepción.

Los dos mensajes que forman el RPC se envían y reciben desde los puertos que corresponden al manejador (mensaje inicial) y a la víctima (mensaje de respuesta). El puerto del manejador se registra como el puerto de excepción para el hilo o tarea víctima; el núcleo consulta este registro cuando ocurre una excepción. El puerto de respuesta se especifica en el mensaje inicial; para las excepciones de hardware, el núcleo proporciona el puerto de respuesta y los cachés que utilizarán los hilos.

3.2 NeXTSTEP

NeXTSTEP es un ambiente de desarrollo orientado a objetos basado e UNIX con una interfaz gráfica, que corre sobre el micronúcleo de Mach, como se aprecia en la figura 4. Sus componentes principales son:

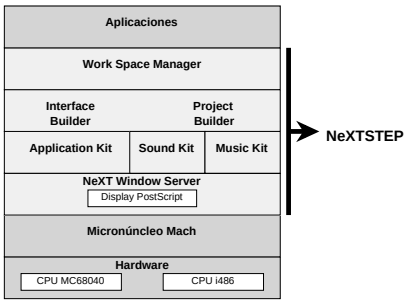

Figura 4: El ambiente de desarrollo NeXTSTEP.

- *NeXT Window Server*. Este es un proceso de bajo nivel que utiliza el *Application Kit* para el manejo de ventanas y enviar los eventos del usuario (acciones del ratón y del teclado) a una aplicación. Este servidor de ventanas incluye un interprete de PostScript (Display PostScript) que se utiliza para dibujar textos y graficas en el video o en papel impreso.
- Bibliotecas de objetos *Application Kit, Sound Kit* y *Music Kit* se utilizan para desarrollar aplicaciones en el ambiente NeXTSTEP.
- Las aplicaciones Interface Builder y Project Builder que se utilizan como herramientas para el desarrollo de aplicaciones en NeXTSTEP.
- El Work Space Manager es el intérprete de comandos de NeXTSTEP, esta aplicación es un *shell* gráfico.

Como se aprecia en la figura 4, las aplicaciones se ejecutan sobre el *Work Space Manager*. Las ventanas son el medio principal de comunicación entre el usuario y una aplicación, para esto el *Window Server* maneja esta comunicación con dos acciones principales: dibujar imágenes en el video y enviar los eventos generados por el usuario a la aplicación. Para dibujar las imágenes se apoya en el interprete *Display PostCript*. Para manejar los eventos que el usuario manda a la aplicación se apoya en el concepto "*ciclo del evento*", en el cual el usuario genera un evento (que el Window Server manda a una aplicación), cuando la aplicación recibe el evento, lo maneja con los objetos del *Application Kit* o con su propio código, genera una acción en el video, y se espera a atender el siguiente evento (como se muestra en la figura 5).

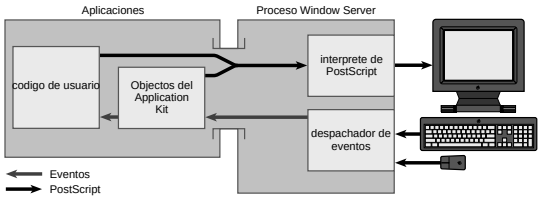

Figura 5: El ciclo del evento.

4 El sistema operativo DOS.

DOS es un sistema operativo modular que consiste de múltiples componentes con funciones especiales cada uno. Cuando DOS se carga a memoria, muchos de estos componentes se mueven, se ajustan o se eliminan. Sin embargo, cuando DOS está en ejecución se puede considerar como una entidad estática y sus componentes son predecibles y fáciles de estudiar. Además, DOS está muy ligado con la arquitectura de la IBM-PC —microcomputadora basada en el procesador de intel 8088— y en las PCs basadas en procesador 8086. Por este motivo DOS trabaja únicamente en un megabyte de memoria, utilizando la forma segmentación como parte de su manejo de memoria. Las áreas principales de la memoria DOS (ver figura 6) son: la parte del vector de

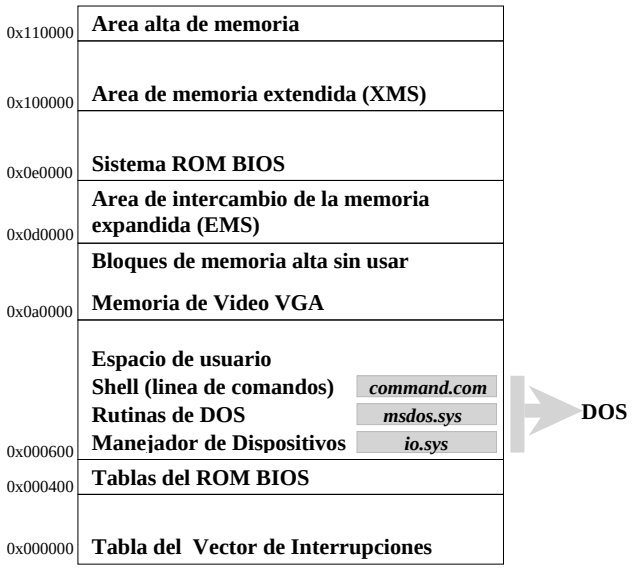

Figura 6: Organización de la memoria de DOS.

interrupciones, las tablas del ROM BIOS, y la memoria de video (arriba de la dirección $0xA0000$). El vector de interrupciones contiene las direcciones de los manejadores de interrupción. Las tablas del ROM BIOS se utilizan para información del estado de varios dispositivos.

DOS no es un sistema operativo multitareas, por lo que si un programa crea un programa hijo el proceso padre queda inactivo y el hijo tendrá todo el control del sistema hasta que termine su ejecución, y el programa padre pueda continuar.

El núcleo de DOS proporciona las funciones que tienen los sistemas operativos tradicionales y algunas funciones propias: manejo de archivos y directorios, manejo de dispositivos de carácter de entrada y salida, soporte de fecha y hora, manejo de memoria, manejo de tarea y ambiente, y configuración de un país específico.

Los programas DOS utilizan dos módulos para soporte del sistema: el ROM BIOS y el núcleo de DOS, los cuales se accesan a través de la instrucción de interrupción por software *INT*. El BIOS es un conjunto de rutinas ROM que soporta los llamados al sistema de dispositivos es-

pecíficos. El núcleo de DOS tiene el soporte de los servicios del SO. El sistema operativo DOS reserva las interrupciones *0x20* hasta *0x3f* para uso propio. La interrupción $0x21$ es la fuente principal para el manejo de los recursos de DOS.

5 Los modos de trabajo del procesador i486

Actualmente los procesadores de la familia de INTEL 80x86 (a partir del 80386) ofrecen tres modos de trabajo: protegido, real y virtual 8086, tanto para facilitar el diseño de sistemas operativos multitareas como para mantener la compatibilidad de ejecución de los programas del 8086/8088 ([3], [5] y [6]). Dos de estos modos de trabajo se utilizan para emular al 8086 (en diferentes formas) y otro es para que el procesador trabaje con sus características nativas:

Modo Protegido. El modo protegido del i486 se considera el modo nativo del procesador, en este modo el procesador trabaja con todas las características de su diseño, como el manejo de caches, manejo de lineas de dirección, manejo de memoria virtual, registros de 32 bits, entre otras cosas. Hablar del modo protegido del i486 es hablar en sí de la forma de trabajo de este procesador.

Modo Real. El modo real del microprocesador i486 puede ejecutar programas escritos para 8086, 8088, 80186, 80188, 80286, 80386 y 80486, o para programas escritos para 80286, 80386 y 80486 en modo real.

La arquitectura del i486 en modo real es muy similar a sus procesadores previos. Para un programador, el 486 en modo real se presenta como un 8086 de alta velocidad con extensiones al conjunto de instrucciones y registros.

Modo Virtual 8086. A partir del INTEL i386, los procesadores soportan la ejecución de uno o más programas 8086, 80186 ó 80188 en el modo protegido. Un programa 8086 corre en este ambiente como parte de una tarea virtual 8086. Una tarea virtual 8086 tiene la ventaja del soporte multitareas que ofrece el hardware en el modo protegido. Podemos tener múltiples tareas virtuales 8086 ejecutando programas 8086 y, además, las tareas virtuales 8086 pueden correr en un ambiente multitareas con otras tareas, en modo protegido, del procesador.

6 Ejecución de código del procesador 8086 en modo virtual del i486

Cuando el procesador trabaja en este modo, realiza una emulación del 8086. Un procesador trabaja en modo virtual cuando se enciende el bit VM (Virtual Machine) del registro de banderas EFLAGS. El procesador revisa esta bandera bajo dos condiciones generales:

1. Cuando se cargan registros de segmentos, para saber si se utilizará el mapeo de direcciones al estilo 8086.
2. Cuando decodifica instrucciones, para determinar que instrucciones son susceptibles al IOPL[2].

Excepto por las instrucciones que producen excepciones en el modo virtual 8086, las demás instrucciones se ejecutan en forma similar al modo protegido.

El cambio de modo virtual a modo protegido se realiza cuando el programa 8086 produce una excepción o una interrupción o cuando se hace un cambio de tarea, figura 7.

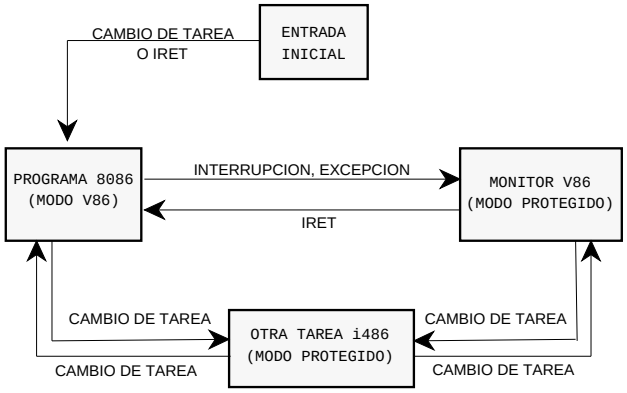

Figura 7: Cambio del modos de trabajo virtual a protegido y viceversa.

Para poder ejecutar un programa 8086 en modo virtual 8086 necesita de un proceso, al que llamaremos monitor virtual 8086, y de los servicios del sistema operativo. El monitor virtual 8086 es código del procesador en modo protegido que corre en el nivel de privilegio 0 (mayor privilegio), y consiste de procedimientos de inicialización y manejo de excepciones.

7 DOS como aplicacion de NeXTSTEP

La parte central para la emulación de DOS para que funcione como una aplicación en NeXTSTEP, es la creación de de una tarea que puede crear un hilo virtual.

La aplicación de DOS puede descomponerse en tres partes principales:

- Soporte del núcleo de Mach. Este soporte maneja el hilo que se ejecuta en modo V86 y las fallas que generan.
- Soporte de NeXTSTEP para el manejo de la interfaz grafica de usuario y de los eventos del usuario para la aplicación DOS.
- Tarea emulador DOS. Esta tarea proporciona a DOS y un soporte de los llamados al BIOS.

7.1 Soporte de Mach para DOS.

El núcleo de Mach debe proporcionar el manejo para los hilos en modo V86, y la interfaz entre el hilo V86 y su tarea de emulación. Específicamente, el núcleo debe proporcionar:

[2] I/O Privilege Level

- Creación del hilo V86 y su mantenimiento a través de los llamados a Mach.
- Creación de la aplicación Emulador asociada al hilo V86.
- Manejo de las fallas de protección general, o excepciones, ocasionadas por el hilo V86.
- Simulación externa[3] de interrupciones en el hilo V86.

7.2 Soporte de NeXTSTEP para DOS

NeXTSTEP proporciona el manejo de ventanas y objetos gráficos, y objetos para interaccionar con el ambiente. Específicamente NeXTSTEP deberá proporcionar:

- La creación de la aplicación DOS.
- El manejo de los eventos de ratón y de teclado.
- Manejo de la salida de video.

Como la comunicación entre el usuario y la aplicación se realiza a través de las ventanas y ,especificamente, de sus objetos gráficos (como son botones, campos de texto, y palancas, entre otros), se deberá crear un objeto gráfico que se encarge de esta tarea, al que denominaremos objeto *VideoVGA*, como se aprecia en la figura 8.

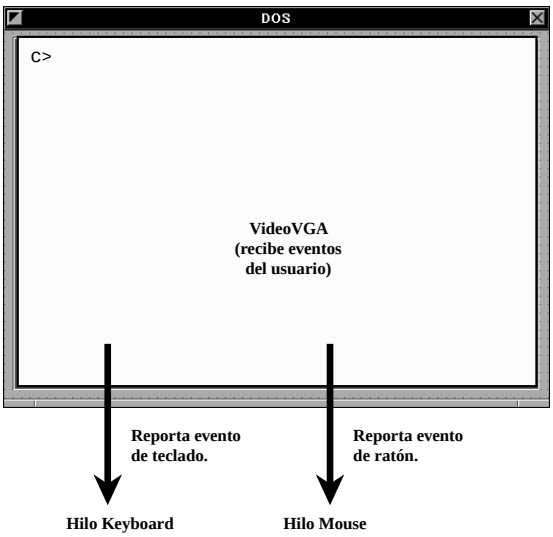

Figura 8: Interfaz de la aplicación DOS.

El objeto *VideoVGA* se encargará de hacer la simulación de un dispositivo VGA y de atender los eventos de ratón y de teclado que genere el usuario.

El video

El objeto *VideoVGA* simulará el comportamiento de un monitor VGA, por lo que deberá mantener una comunicación con el proceso *monitor* y la tarea *emulador* para simular el manejo del video cuando se realice por medio de la instrucción de interrupción por software *INT*, o cuando se lleve a cabo por el DMA.

[3]Llamamos simulación externa al código que se ejecuta por otro hilo en modo protegido.

Manejo de eventos del usuario

El *Window Server* enviará al objeto *VideoVGA* los eventos generados por el usuario. Cuando el objeto *VideoVGA* reciba la notificación de los eventos se encargará de enviarlos a los hilos *Keyboard* y *mouse* para que sean procesados adecuadamente.

7.3 La tarea Emulador

Debido a que los servicios que ofrece un sistema operativo se pueden distribuir en un conjunto de servidores, consideramos a la aplicación Emulador como una aplicación multihilos que soportan las emulación de DOS, los llamados a BIOS y que proporcionen la entrada al hilo V86, como se plantea en [1] (figura 9). Estos hilos en la tarea monitor deben ser:

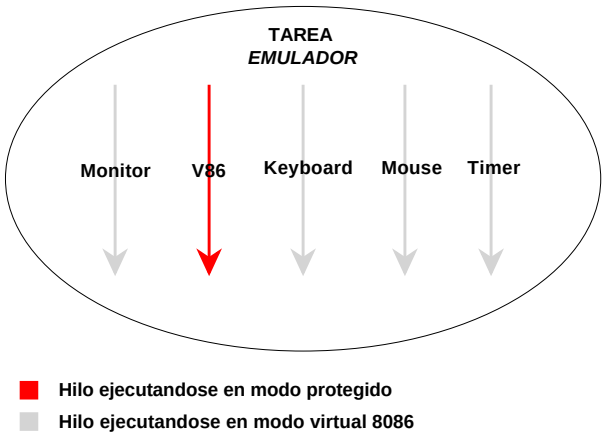

Figura 9: Hilos de la aplicación Emulador.

Hilo Monitor. Proporciona la emulación para los llamados a DOS y BIOS, al igual que realiza la simulación del conjunto de instrucciones privilegiadas.

Hilo V86. Es el único hilo que opera en modo virtual 8086. Ejecuta las aplicaciones de DOS y del núcleo que generan los errores de protección general. Estos errores son provocados por las instrucciones privilegiadas que atrapa el núcleo.

Hilo Timer. Genera las dos interrupciones de reloj de DOS: la interrupción de fecha y hora, y la interrupción que genera el pulso de reloj.

Hilo Mouse. Emula el manejador de dispositivo de ratón o un ratón conectado a un puerto serie. Este hilo se activará cada vez que el objeto *VideoVGA* le notifique que ha ocurrido un evento de ratón. Emula el manejador de dispositivo de ratón o un ratón conectado a un puerto serie.

Hilo Keyboard. Recoger los datos emitidos por el teclado y se los transmite al hilo V86. El hilo Keyboard se activará cada vez que el objeto *VideoVGA* le notifique que ha ocurrido un evento de teclado.

8 El hilo virtual V86

La parte central para un soporte de DOS en Mach es trabajar en el modo virtual 8086 del procesador i486, esto es, el procesador i486 emulará la forma de trabajo del 8086. Este soporte se apoya en la forma en que se realizan los llamados al sistema DOS, la organización física de la memoria de una máquina DOS, y la forma en que Mach maneja las excepciones generadas por las instrucciones especiales.

Para ejecutar un programa 8086 en Mach, se necesita un proceso monitor, para soporte del manejo de excepciones, un área de memoria para realizar el mapeo de la memoria de una máquina DOS, a la que denominaremos tarea V86, y un hilo que ejecute el programa 8086 en modo virtual, al que llamaremos hilo virtual V86. A la tarea V86 y al hilo virtual V86 lo consideramos como un proceso UNIX al que llamaremos proceso V86.

El proceso monitor se encarga de inicializar el proceso V86, esto es, crear la tarea V86 y preparar el primer megabyte para crear la máquina DOS, cargar el programa 8086, y crear e inicializar el hilo V86 para que ejecute el programa 8086. El hilo V86 se ejecuta en el primer megabyte de memoria de una tarea Mach e intentará ejecutar instrucciones especiales que afectan al sistema en modo protegido. Los pasos para crear el hilo virtual V86 son:

1. Limpiar el primer megabyte de memoria de la tarea V86 y asignarle el código del programa 8086. Este paso se realiza con ayuda del llamado a Mach *task_create* y las primitivas de lectura y escritura a la memoria virtual .
2. Crear e inicializar el hilo V86. Esta operación se realiza con los llamados a Mach *thread_create*, para crear el hilo V86; *thread_get_state*, para obtener el estado inicial del hilo V86; y *thread_set_state*, para asignarle los valores necesarios para inicializar un hilo en modo virtual 8086. El llamado *thread_get_state* regresa la estructura del estado del hilo para poder inicializar los registros que se cargarán en el procesador. Una vez obtenidos los valores del hilo, se enciende la bandera VM (Virtual Machine) del registro EFLAGS —para declarar que el hilo V86 efectivamente se ejecutará en modo virtual 8086—, darle valores a los registros SS y SP para asignarle una pila, e indicarle la dirección CS:IP del programa 8086 que se ejecutará.
3. Crear el puerto que recibirá del núcleo los mensajes de excepción generados por la ejecución del hilo V86. Este puerto se crea en el proceso monitor, ya que este proceso es el encargado de manejar las excepciones, aunque se asocia al hilo V86, que a su vez, se encuentra asociado a la tarea V86.

Gracias a las facilidades que ofrece Mach, podemos obtener el estado del hilo V86 en su inicialización, y modificarlo para indicarle que se ejecutará en el modo virtual 8086. Estas facilidades no se encuentran en otras plataformas como UNIX o LINUX. Una vez que el proceso monitor a creado e inicializado la tarea V86 y el hilo V86, pondrá en ejecución al hilo V86 y esperará a que éste genere excepciones. El núcleo informará al proceso monitor, a través del puerto de excepción, que el hilo V86 a ocasionado una excepción para que la maneje.

Cuando el hilo virtual V86 genera una excepción, Mach suspende la ejecución de este hilo y notifica la excepción al puerto asociado para este propósito. Cuando el proceso monitor recibe este mensaje primero verifica el tipo de excepción que se ha generado —las excepciones pueden ser de tipo aritmético, de protección o señales de hardware— y dependiendo de esto se procede a hacer una simulación, si es de protección o de hardware, o una corrección de datos, si es aritmético. Cuando se ha atendido la excepción el proceso monitor notifica a Mach para que se reanude la ejecución del hilo virtual V86, como se muestra en la figura 10. El manejo de excepciones requiere que el proceso monitor realice operaciones de lectura y escritura a la tarea V86, esto se debe a que en la simulación de una instrucción es necesario conocer cual instrucción especial está generando la excepción — Mach notifica al monitor que ha ocurrido una excepción de protección, más no le indica que instrucción lo ha ocasionado—, y dependiendo de esta instrucción, se procede a modificar el espacio de direcciones de la tarea V86 y el estado del hilo V86, tal y como lo haría el procesador si ejecutara la instrucción especial.

El manejo de excepciones requiere de una gran comunicación del proceso monitor con la tarea V86, lo que puede ser un grave inconveniente si no se plantea en forma adecuada. Se probaron diferentes modelos de comunicación entre el proceso monitor y la tarea V86. El diseño que resulto óptimo consiste en compartir la memoria de la tarea V86 con el proceso monitor haciendo un mapeo de direcciones, tal que el primer megabyte de memoria de la tarea V86 se accese con algún megabyte que tenga disponible el proceso monitor (figura 11).

El proceso V86 y proceso monitor se ejecutan en diferentes modos de operación (virtual 8086 y protegido respectivamente), y Mach se encarga de realizar la sincronización entre ambos procesos para realizar la ejecución del programa 8086.

9 La aplicación *VM*

Aprovechando las facilidades de NeXTSTEP para desarrollar aplicaciones, se creó la aplicación gráfica, orientada a objetos y multihilos *VM* para monitorear la ejecución del hilo virtual V86. Esta aplicación incluye:

- Módulo para cargar un programa 8086 con formato *COM* a una tarea V86.

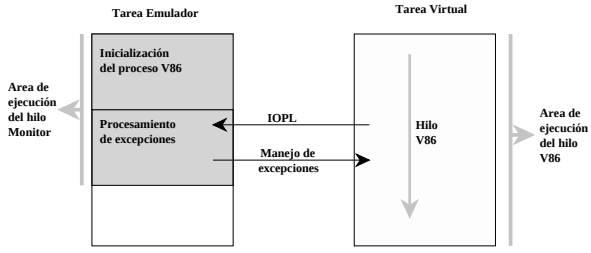

Figura 10: Manejo de excepciones del hilo V86.

- Desensamblador para 8086.
- Módulo para visualizar el estado de la memoria.
- Módulo para visualizar el tipo de instrucción especial que genere un error y el estado del hilo V86.
- Módulo para comunicar los diferentes hilos con los objetos gráficos de la GUI de la aplicación *VM*.

La interfaz de la aplicación, ver figura 12, despliega en una ventana el programa 8086 que se va a ejecutar, el estado de la memoria donde se ejecuta el programa 8086, el tipo de instrucción que genera una excepción especial, la cual se va a asimular, y el estado del hilo V86 cuando ocurre una excepción generada por alguna instrucción especial.

Para este soporte la aplicación corre con cuatro hilos, tres de ellos ejecutándose en modo protegido y uno en modo virtual 8086, como se aprecia en la figura 13:

Hilo V86 Este hilo se ejecuta en modo virtual 8086.

Hilo Monitor Este hilo se encarga de inicializar la tarea V86 y de manejar las excepciones que genere el hilo V86. También se encarga de notificar al hilo GUI para desplegar el tipo de instrucción especial que generó la excepción y el estado del procesador.

Hilo Memoria Se encarga de mandar al hilo GUI el estado de la memoria para que se despliegue en la GUI de la aplicación.

Hilo GUI Este hilo se encarga de manejar la GUI de la aplicación y mostrar la salida de los diferentes hilos de ejecución.

La aplicación *VM* se encarga de construir a la tarea V86, de inicializar el hilo virtual V86, y ejecutar al proceso monitor cada vez que se carga un programa 8086.

9.1 Diseño de la aplicación *VM*

Los objetos que se ejecutan en la aplicación *VM* pertenecen a dos clases principales:

VirtualApp Es una subclase de la clase *aplication*. El objeto de esta clase se encarga de manejar la GUI de la aplicación, esto es: manejar la carga de programas 8086, mandar a ejecutarlos, mostrar en los objetos gráficos de la GUI, los datos que le envían los hilos monitor y memoria.

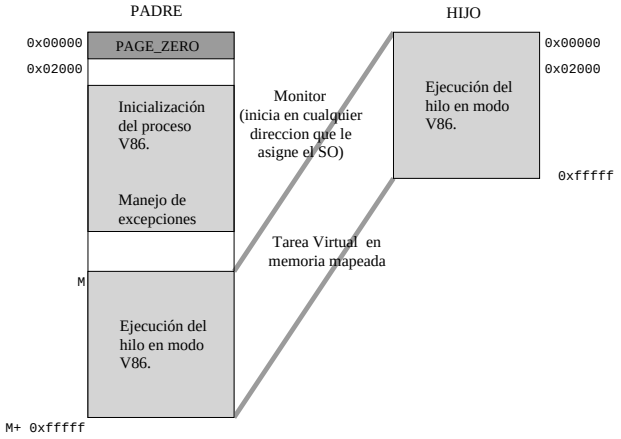

Figura 11: Diseño Padre-Hijo con memoria mapeada

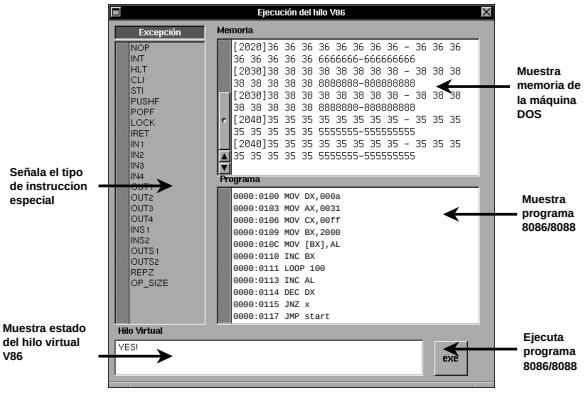

Figura 12: Interfaz gráfica de la aplicación *VM*.

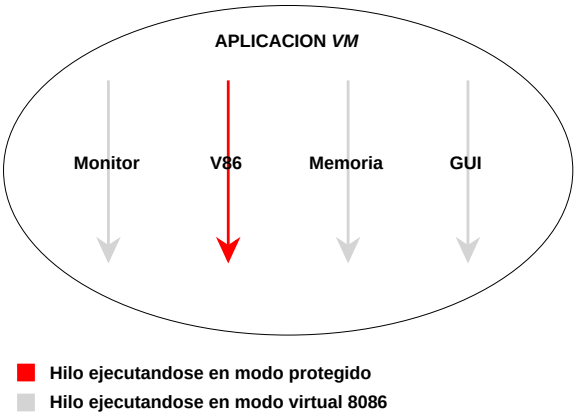

Figura 13: Hilos de la aplicación *VM*.

ThreadEngine Es una subclase de la clase raíz *Object*. Los objetos de esta clase generan hilos de Mach que trabajan en modo protegido. Los hilos monitor y Memoria son objetos creados de esta clase.

Los principales objetos que se crean en la aplicación son:

- Objeto aplicación generado de la clase *VirtualApp*, el hilo que ejecuta a este objeto le llamamos hilo GUI.
- Objeto monitor, este objeto pertenece a la clase *ThreadEngine* y se encarga de ejecutar el proceso monitor, que a su vez se encarga de crear al proceso V86 y atender sus excepciones.
- Objeto memoria, este objeto pertenece a la clase *ThreadEngine* y se encarga de leer la memoria donde se ejecuta el hilo V86 cada determinado tiempo, y enviarle un mensaje al objeto GUI para que la despliegue en un objeto gráfico.

9.2 Ejecución de la aplicación *VM*

La aplicación *VM* se ha desarrollado para ejecutar programas 8086 con formato *COM*. Cuando se ejecuta la

aplicación *VM*, NeXTSTEP crea el objeto NXApp, lo pone a ejecutar y se muestra la interfaz de usuario de la aplición.

Cuando se carga un programa 8086, el hilo GUI se encarga de crear los objetos monitor y memoria, y los pone en ejecución. Esto se logra gracias a que ambos objetos son hilos. Cuando se ejecuta el objeto monitor se encarga de crear e inicializar al proceso V86, cargando el programa 8086 en una localidad de memoria de la tarea V86 y le asignándole a los registros CS e IP del hilo V86 la dirección inicial donde se cargó este programa.

Los objetos que se ejecutan son el objeto GUI, el objeto memoria y el objeto monitor y se comunican entre ellos a través de mensajes. Los objetos monitor y memoria se comunican con la tarea V86 compartiendo el área de memoria de la tarea V86, ver figura 14.

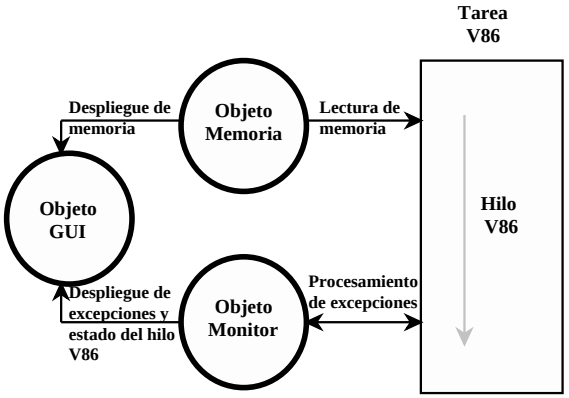

Figura 14: Comunicación entre la aplicación *VM*.

Cuando se esta ejecutando el proceso V86, el hilo GUI espera a que el hilo monitor y el hilo de memoria le envíen mensajes que se mostrarán en interfaz gráfica. El hilo de memoria se activa cada determinado tiempo y se encarga de leer una región de la memoria de la tarea V86 y la envía al hilo GUI para que se muestre al usuario.

El hilo V86 y el objeto monitor se comunican para el manejo de excepciones. Cuando el objeto monitor procesa una excepción envía el tipo de excepción y el estado del hilo V86 al objeto GUI para que se muestre al usuario.

10 Resultados y conclusiones.

Se logró ejecutar programas sencillos 8086 en Mach, versión 3.3 para i486, y se desarrolló el soporte para ejecutar un programa 8086 en el ambiente de NeXTSTEP. Aunque la simulación y la emulación difieren en principio, el soporte que se desarrolló se apoya en la emulación del 8086 por el 80486, y en las técnicas de simulación para el manejo de excepciones. Se ha probado la aplicación *VM* con programas 8086 con formato COM, que realizan su entrada y salida únicamente con la memoria.

Como los programas 8086 se ejecutan en una plataforma de hardware que reconoce sus instrucciones, su ejecución se realiza a través de la emulación del 8086, esto es, el procesador i486 se comporta como un procesador 8086, sin embargo, para obtener la ejecución de las instrucciones privilegiadas, se realiza la simulación de cada una de éstas, esto es, el proceso monitor realiza los cambios al espacio de memoria la tarea V86 y a los registros del hilo V86, como si en realidad la instrucción especial se hubiese ejecutado.

Los problemas más fuertes a los que nos enfrentamos fue en la comunicación del proceso monitor con la tarea V86 para realizar la simulación de las instrucciones especiales.

Podemos concluir que es factible crear todo el emulador de una máquina 8086 evitando hacer modificaciones a DOS. Además utilizando las facilidades de que ofrece NeXTSTEP, podemos concentrarnos en escribir un objeto que simule un monitor VGA y que se encarge procesar los eventos de ratón y de teclado para la aplicación DOS.

Referencias

[1] Gerald Malan, Richard Rashid, David Golub, and Robert Baron; "DOS as a Mach 3.0 Aplication"; School of Computer Science; Carnegie Mellon University.

[2] Mendel Rosenblum and Mani Varadarajan, "SimOS: A Fast Operating System Simulation Environment"; Technical Report:CSL-TR-94-631; Computer Systems Laboratory, Department of Electrical Engineering and Computer Science; Standford University; July 1994.

[3] "386^{tm}SX Microprocessor Programmers's Reference Manual";INTEL Osborne McGraw-Hill; 1989.

[4] "The 8086 Family Users's Manual"; INTEL; October 1979.

[5] William J. Claff, "MOVING FROM THE 8088 TO THE 80286, Important differences you need to know to make your programs transportable", Fall 1985 BYTE, Inside the IBM PCs, pags. 93-101.

[6] L. Brett Glass; "PROTECTED MODE", *Under de Hood*; BYTE; December 1989, Pags. 377-384.

[7] Richard Rashid, Robert Baron, Alessandro Fornin, David Gulob, Michael Jones, Daniel Julin, Douglas Orr, Richard Sanzi; "Mach: A Fundation for Operating Systems, A Position Paper"; School of Computer Science; Carnegie Mellon University.

[8] Keith Loepere, "Mach 3 Kernel Principles"; Open Software Fundation and Carnegie Mellon University; NORMA-MK12; July 15, 1992.

[9] NeXT Computer, Inc., "NeXT Manuals", 1995.