Trabajo Práctico n° 3

Sistemas Operativos

- Alumnos
  - Ballesty, Pablo Andrés 49359
  - Lezica, Santiago 49147
  - Magnorsky, Alejandro 50272
  - Mata Suárez, Andrés Ricardo 50143
  - Merchante, Mariano 50094
  - Pose, Jimena Belén 49015

- Profesores
  - Etchegoyen, Hugo Eduardo
  - Carlucci, Bruno Luciano Nicolás

- Fecha de Entrega
  - 29/11/10

- 2010 -

Índice

1. Introducción

1.1. Objetivo

Continuar el Multitasker del Trabajo Práctico 2, agregándole manejo de disco, basado en el sistema de manejo de disco de MINIX.

1.2. Enunciado

El trabajo consta de la realización de un manejador de disco símil al de MINIX, para el sistema multitasker creado en el TP2. El mismo deberá poder leer y escribir bytes a disco, pero no requerirá un filesystem. La información escrita a disco, naturalmente debe poder persistir cuando la máquina se apague o se reinicie.

1.3. Programas de prueba

Además del sistema a disco, los alumnos deberán desarrollar en assembler un programa pequeño que carezca de stack y sea independiente de contexto. Los alumnos deberán poner este programa en el disco en una zona conocida y agregar un comando usuario que acceda a esta zona conocida de disco, levante el programa en una página nueva y cree un nuevo proceso que ejecute ese programa. Como este programa es independiente del sistema operativo, deberían poder ejecutarlo protegiendo el sistema operativo completo mientras el mismo se ejecuta y deberían poder proteger todas sus zonas (incluyendo código) cuando otro programa se ejecuta.

1.4. Material a entregar

Cada grupo deberá entregar los fuentes, una imagen booteable con el bootloader y el multitasker y los datos pertinentes cargados a disco. Además deberán entregar un informe impreso detallando las decisiones respecto a los items que quedaron a elección del grupo, problemas presentados y la solución de los mismos durante la realización del trabajo.

1.5. Integrantes

El trabajo puede hacerse en grupos de seis integrantes. Se evaluará la funcionalidad de la aplicación así como el estilo del código y la calidad del informe entregado.

1.6. Consideraciones

Todo punto no explícito en este documento podrá ser interpretado a convenieniencia del alumno, siempre dentro de los márgenes del sentido común. Ante la duda, consultar a los docentes o enviar un mail al mail de la cátedra.

1.7. Fecha de entrega

La fecha de entrega del trabajo práctico es el Lunes 29 de Noviembre a las 16:00.

2. Desarrollo

2.1. Funcionamiento del driver

Para leer y escribir del disco se desarrollaron las funciones _read y _write. Cada una lee o escribe un sector en el disco, recibiendo un cierto offset y una cantidad determinada de bytes. Para una mayor funcionalidad se implementaron las funciones read y write, las cuales dan la posibilidad de escribir o leer del disco más de un sector. Para poder leer o escribir en el disco hay que enviar un comando al disco (LBA_READ o LBA_WRITE respectivamente), así como la cantidad de sectores y el sector inicial a los registros correspondientes del disco. Una vez que se termina el comando, se debe leer o escribir la información del registro 0, la cual tiene un tamaño de 2 bytes (1 word). Para lograr procesar los datos obtenidos se creó la función translateBytes, con la cual se logra transformar una word en un arreglo de 2 bytes.

2.2. Problemas encontrados y sus respectivas soluciones

Comenzamos intentando utilizar los archivos at_wini.c y driver.c de Minix 2.0.4. En un principio incorporamos funciones poco a poco, encontrando el problema que a medida que agregábamos una función a nuestro sistema operativo nos veíamos forzados a importar mucho contenido fuera de lo que era el manejo del disco (como constantes, macros, funciones extras, etc). Otro problema encontrado fue que nos vimos forzados a incorporar los métodos de IPC análogos a los de Minix (ya que los IPCs con los que contábamos eran muy diferentes), los procesos de Minix (la estructura de un proceso) y el manejo de interrupciones.

Intentando resolver este problema nos encontramos con que esta versión de Minix usa un tipo de dato llamado u64, que sirve para el manejo de disco en sistemas de 64 bits, lo que dificultaba aún más la incorporación del driver de Minix a nuestro sistema operativo.

Como consecuencia de lo antes mencionado, decidimos utilizar la versión más básica de Minix, la 1.1. Sin embargo, si bien ya no teníamos el problema de los 64 bits, los otros problemas seguían presentes. Finalmente optamos por investigar por otros medios el funcionamiento del driver de disco ya que las incompatibilidades entre ambos sistemas operativos era muy grande. Es por eso que recomenzamos el trabajo, pero esta vez armando nuestro propio driver basándonos en las especificaciones del disco ATA (El pdf utilizado se encuentra en la carpeta docs entregada bajo el nombre de "Manual_ATA_ATAPI.pdf").

Utilizamos el modo LBA (Logical block addressing), el cual sirve para especificar la localización de los bloques de datos en el disco duro. Los bloques lógicos en nuestro sistema operativo son de 512 bytes cada uno. Este sistema nos permite tratar al disco como si fuera un vector haciendo que la lectura y escritura sea rápida y sencilla.

Tanto la lectura como la escritura en el disco son System Calls, asegurándonos de que nadie interrumpa dichas operaciones ya que están protegidas por cli y sti.

2.3. Programas de prueba

2.3.1. Programa en el disco

Hicimos un programa que llena la pantalla con el caracter “a” de color rojo.

En un principio intentamos compilarlo externamente y copiar en el primer sector del disco el código objeto obtenido (utilizando el programa de edición hexedit). Cuando ejecutábamos el comando programDisk, alojábamos dicho código objeto en una página en memoria, para luego hacer un call a la primer posición de memoria de dicha página. Ejecutando programDisk obteníamos errores de protección. El error surgía de que el código objeto obtenido externamente presentaba diferencias al código objeto del mismo programa pero compilado internamente. Para solucionarlo, imprimimos todos los bytes que componían el código objeto compilado internamente y los copiamos tal cual al primer sector de disco.

2.3.2. File system

Para ejemplificar el trabajo realizado, decidimos desarrollar un sistema de archivos minimal. Básicamente está conformado por un árbol sencillo de archivos, con la raiz ” %”, debido a que /”no estaba del todo mapeada en el teclado. No hay distinción entre archivo y directorio; todo archivo puede poseer una cantidad limitada de hijos. Si bien esto no es lo óptimo, es fácil de resolver agregando un bit en el encabezado del archivo en disco. La persistencia de los archivos se logra guardando en disco un encabezado de 512 bytes en algun sector vacío, y el contenido mismo del archivo en n sectores posteriores al del encabezado. Los sectores de cada archivo son contiguos; es decir, el tamaño del archivo se fija cuando se crea, y no puede crecer. Si bien esto es una limitación importante, también es sencillo de resolver, utilizando listas de sectores, aunque supera el objetivo de esta demostración. La presencia de los sectores se resuelve con un mapa de bits que mapea 10 megabytes (el tamaño del disco que usamos), en el cual el n-ésimo bit dice si el sector n esta ocupado o no. Este mapa de bits se guarda en disco, y se intenta levantar al inicio del sistema operativo; si no se encuentra, se crea un mapa nuevo y se construye un árbol inicial. Asimismo, si encuentra el mapa de bits, lo levanta, e intenta cargar archivos desde todos los sectores ocupados. Si no llegara a encontrar un archivo, simplemente saltea al próximo sector ocupado y libera la información. Luego de cargar todos los archivos, reconstruye el árbol de archivos, basándose en la información persistida en el disco. Cada archivo guarda en el disco su nombre, su tamaño y el sector de su padre (el directorio en el que reside), además de dos strings que definen si es un archivo existente o no. Gracias a este método, es sencillo eliminar archivos y directorios; simplemente se corrompe el string del encabezado del archivo. Incluso se podrían recuperar archivos eliminados, ya que se utiliza un encabezado distinto para archivos eliminados.

Además del sistema de archivos se incluyeron varios programas: cd, ls, rm, mkdir, edit, tree, pwd [-a], entre otros. Estos programas sirven para moverse dentro del árbol y poder ver, crear, editar o borrar archivos y directorios.

3. Conclusión

Intentar utilizar el driver de Minix en nuestro sistema operativo fue contraproductivo, ya que diferían demasiado en cosas críticas. Haberlo programado nosotros mismos fue más didáctico y además tuvimos mayor control de lo que hacía el driver. Un punto imprescindible para la realización del trabajo práctico fue la investigación realizada para encontrar la forma específica de interactuar con el controlador de disco.

Cabe mencionar que nuestro driver de disco es compatible únicamente con los dispositivos AT y ATA. Con respecto al sistema de archivos, hubo varias cosas que nos hubiera gustado poder completar, como lista de sectores y directorios bien implementados. También, no alcanzó el tiempo para extender la funcionalidad del sistema como nos hubiera gustado.