Evaluación 2 Sistemas Operativos I

Análisis de carga

Puntaje máximo: 20 puntos

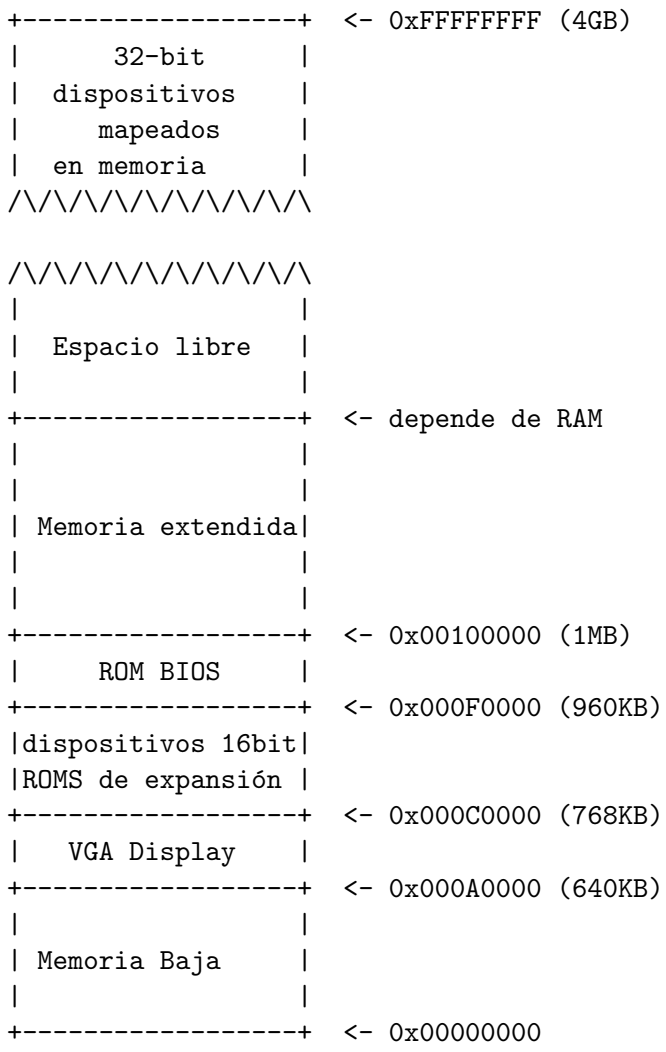
En todo lo que sigue, "sistema" siempre se refiere al sistema operativo pintos a menos que se especifique de otra manera.

1. Introducción

1.1. La carga[1] del sistema operativo

El proceso de cargar un sistema en memoria para posteriormente ejecutarlo luego de encender una PC se conoce como carga[2]. El sistema cargará otros programas, como lo es una interfaz de línea de comando, para que el usuario tenga una plataforma para hacer su trabajo. Hay dos grandes elementos responsables de construir el camino para hacer la carga: BIOS (Basic I/O System) y el cargador[3]. El hardware de la PC está diseñado para asegurarse que el BIOS siempre tenga control de la máquina luego de que esta se encienda. El BIOS[4] iniciará haciendo algunas pruebas y código de arranque, por ejemplo, asegurarse de saber cuánta memoria disponible existe y activar la tarjeta de video. Luego de esta iniciación, el BIOS intentará encontrar un dispositivo cargable de alguna ubicación apropiada tal como lo es un disco duro, USB o la red. Luego, el BIOS pasará control de la máquina al cargador quien finalmente cargará al sistema operativo.
A pesar de que el BIOS y el cargador tienen tareas muy importantes, cuentan con muy pocos recursos para trabajar. El cargador de IA-32 debe caber en 512 bytes de una partición del disco duro, específicamente en su primer sector. Los últimos 2 bytes se utilizan para determinar que en efecto es un cargador. Para poner un cargador en el Master Boot Record (MBR), éste debe caber en un espacio de 436 bytes. Además, dado que el BIOS y el cargador corren realmente sobre hardware, no hay bibliotecas estándares para llamar funciones tales como *sys_read* o *sys_write*. Muchas funcionalidades necesitan ser implementadas desde cero. Por ejemplo, leer contenido de un disco es relativamente fácil si nos encontramos dentro de un sistema operativo, pues tenemos acceso a llamadas al sistema que nos facilitan esta tarea. Sin embargo, en un cargador, tenemos que trabajar directamente con las rutinas complicadas del hardware. Un resultado de lo anterior es que los cargadores por lo general se escriben en ensamblador, porque incluso un compilador de C incluiría demasiado exceso.
Para entender en más profundidad este reto, es útil considerar el espacio físico de direcciones de la PC, el cual está hecho para tener la siguiente estructura:

[1] "Bootloading"
[2] "bootstrapping"
[3] "bootloader"
[4] "**B**asic **I**nput **O**utput **S**ystem"



Las primeras PCs, que fueron basadas en el procesador Intel 8088 de 16-bit, sólo eran capaces de direccionar 1MB de memoria física. El espacio de direcciones físicas en una de esas PCs empezaría en 0x00000000 y terminaría en 0x000FFFFF. El área de 640 KB marcada como "Memoria Baja" era la única memoria de acceso aleatorio (RAM) que una PC utilizaría. De hecho, las primeras PCs sólo podían ser configuradas con 16 KB, 32 KB o 64 KB de RAM.
El área de 384 KB de 0x000A0000 a 0x000FFFFF era reservada por el hardware para usos especiales tales como despliegue de video y el firmware almacenado en memoria no volátil. La parte más importante de esta área reservada es el BIOS, que ocupa una región de 64KB de 0x000F0000 a 0x000FFFFF. En las primeras PCs el BIOS era mantenido en memoria realmente de sólo lectura (ROM), pero PCs más recientes almacenan el BIOS en memoria flash actualizable.
Cuando Intel finalmente rompió la barrera de 1MB con los procesadores 80286 y 80386, que soportaban espacios de memoria físicos de 16MB y 4GB respectivametne, los arquitectos de PCs preservaron la configuración original de 1MB de direcciones físicas bajas para asegurar compatibilidad con software existente. Las PCs más recientes tienen entonces un "hueco" en memoria física, de 0x000A0000 a 0x00100000, dividiendo la RAM en memoria "baja" o "convencional" (los primeros

640KB), y memoria extendida (el resto). Asimismo, cierto espacio en la parte alta del espacio de direcciones de 32-bit, encima de la RAM física, se reserva comúnmente por el BIOS para el uso de dispositivos PCI de 32-bit.

1.2. El cargador

Los discos duros se dividen en regiones de 512 bytes llamadas sectores. Un sector es la unidad de transferencia de data de granularidad más baja: cada lectura o escritura debe ser de tamaño igual a uno o más sectores, y debe estar alineada con los límites de los sectores (no es posible leer empezando desde la mitad de un sector). Si el disco puede ser cargado, el primer sector es llamado el sector de carga (boot sector), puesto que es el lugar donde se ubica el código cargador. Cuando el BIOS encuentra un disco duro cargable, este carga el primer sector (de 512 bytes) en la memoria en el espacio de direcciones físicas de 0x7c00 a 0x7dff, y luego utiliza una instrucción JMP para configurar el CS:IP (puntero a instrucción del segmento de código) a 0000:7c00, efectivamente pasando control al cargador (boot loader).

Los cargadors de la arquitectura IA-32 tienen la indeseable característica de correr en modo de direccionamiento real (también conocido como "modo real"), donde los registros de segmentos se utilizan para computar la dirección de los accesos a memoria de acuerdo a la siguiente fórmula: $16 \times$ segmento + offset. El segmento de código CS se utiliza para la ejecución de instrucciones. Por ejemplo, cuando el BIOS salta (JMP) a 0x0000:7c00, la dirección física correspondiente es $16 \times 0 + 7c00 = 7c00$. Otros registros de segmentos son el segmento de pila (SS), el segmento de data (DS) y el segmento ES también para mover data.

Tenga presente que cada segmento es de 64 KiB. Debido a que los cargadores usualmente tienen que cargar núcleos mayores que 64 KiB, estos deben utilizar los segmentos de registros cuidadosamente.

La carga en Pintos es muy simple en comparación con otros sistemas operativos modernos. El núcleo[5] es de máximo 512 KiB (ó 1024 sectores), y debe ser cargado en la memoria iniciando en la dirección 0x20000. Pintos requiere un tipo de partición específica para el sistema operativo, por lo que el cargador de Pintos debe buscar una partición del tipo apropiado. Esto significa que el cargador de Pintos debe entender cómo utilizar el Master Boot Record (MBR). Afortunadamente es sencillo de entender.

Cuando el cargador encuentra una partición cargable, lee los contenidos de la partición en memoria en la dirección física 128 KB. El núcleo se encuentra en el inicio de la partición, lo que puede ser más grande de lo necesario debido a los límites de alineación de las particiones, por lo que el cargador lee no más de 512 KB (el proceso de compilación de Pintos no producirá núcleos más grande de eso). Leer más data que esta también cruzaría la región de 640 KB a 1 MB que la arquitectura reserva para el hardware y el BIOS, y un BIOS estándar no provee ningún medio para cargar el núcleo en direcciones por encima de 1 MB.

El trabajo final del cargador es extraer el punto de entrada de la imagen del núcleo cargada y transferir el control a este. El punto de entrada no se encuentra en una ubicación predecible, pero la cabecera ELF del núcleo contiene un puntero a este. El cargador extrae este puntro y salta (JMP) a la dirección a la que este apunta. La linea de comandos del núcleo Pintos se almacena en el cargador (usando aproximadamente 128 bytes). El programa pintos realmente modifica una copia del cargador en disco cada vez que ejecuta el núcleo, insertando cualquier argumento de línea de comando que haya sido suplido por el usuario, y luego el núcleo, en tiempo de carga, lee esos argumentos desde el cargador en memoria. No es una solución elegante, pero es simple y efectiva.

[5] "Kernel"

1.3. El núcleo

La última acción del cargador es transferir control al punto de entrada del núcleo, el cual es start() y se ubica en threads/start.S. El trabajo de este código es cambiar del modo real de 16-bit al modo protegido de 32-bits, utilizado por todos los sistemas operativos x86 modernos.
La primera tarea del código de inicio del núcleo es averiguar el tamaño de la memoria, preguntándole al BIOS por este. La función más sencilla del BIOS que hace esta tarea solamente detecta 64MB de RAM, por la que Pintos únicamente soporta esto. Asimismo, el código de inicio necesita iniciar la línea A20, esta es, la línea de dirección del CPU enumerada con el 20. Por razones históricas, las PCs cargan con esta línea de dirección fija en 0, lo que significa que cualquier intento de acceder a memoria más allá del primer Megabyte (2^{20}) será fallido. Pintos quiere acceder a más memoria que 1 MB, y por esta razón necesitamos habilitarla.
Posteriormente, el núcleo configurará una tabla de páginas y configurará el modo protegido y paginación (más detalles en la clase). El último paso es llamar al código escrito en C del núcleo de Pintos.

2. Enunciado

2.1. El depurador[6]

La mayoría de los ejercicios en el enunciado están relacionados con el uso del depurador *gdb*. Naturalmente, es necesario familiarizarse con el uso de éste. Para empezar, verifique que tiene instalada la documentación local de *gdb*. Para ello, teclee en su terminal

```
info gdb
```

Si al ejecutarlo lo que aparece informa que el contenido es de alrededor de 200 líneas, entonces es que *no* lo tiene instalado: lo que está viendo es la "man-page". Para instalarlo, teclee

```
sudo apt install gdb-doc
```

Alternativamente, puede acceder a la documentación en línea en https://www.gnu.org/software/gdb/documentation/ La información definitiva sobre *gdb* es

> Richard Stallman, Roland Pesch, Stan Shebs, et al. Debugging with GDB: The GNU Source-Level Debugger

Probablemente, le resulte más comestible el empleo de una interface *visual*. Puede considerar el uso de DDD. Para instalarlo en Ubuntu

```
sudo apt-get install ddd
```

Advertimos que el uso de interface visual puede provocar algunas distorsiones.

[6] "Debugger"

2.2. Ejercicio 1

Utilice *gdb* para darle seguimiento al BIOS de QEMU para entender cómo una computadora IA-32 carga. Responda las siguientes preguntas:

1. ¿Cuál es la primera instrucción en ejecutar?
2. ¿En qué dirección física se encuentra esa instrucción?
3. ¿Cuál es el propósito de la primera instrucción? ¿Por qué fue situada?
4. ¿Cuáles son las próximas tres instrucciones?

2.3. Ejercicio 2

En este ejercicio se dará seguimiento al cargador de Pintos. Coloque un punto de ruptura[7] en la dirección 0x7c00, que es la dirección donde se carga el sector de carga[8]. Continúe la ejecución hasta llegar a ese punto de ruptura. Lea el código en threads/loader.S, utilizando el código fuente y el archivo de des-ensamblaje en threads/build/loader.asm para dar seguimiento a dónde se encuentra. También, utilicd gdb para des-ensamblar secuencias de instrucciones en el cargador, y comparar el código fuente original del bootloader tanto con la versión des-ensamblada en threads/build/loader.asm y gdb.

Responda las siguientes preguntas:

1. ¿Cómo el cargador lee sectores de disco? En particular, ¿qué interrupción de BIOS es utilizada?
2. ¿Cómo el cargador decide si encuentra el núcleo de Pintos?
3. ¿Qué sucede cuando el cargador no puede encontrar el núcleo de Pintos?
4. ¿En qué punto el cargador le transfiere control al núcleo Pintos?

Luego de que el núcleo Pintos toma control, la configuración inicial se realiza a través de código escrito en ensamblador que se encuentra en threads/start.S. Luego, el núcleo pasa a ejecutar código escrito en C llamando a la función *pintos_init()* que se encuentra en threads/init.c. Coloque un punto de ruptura en *pintos_init()* y luego continúe dando seguimiento al código de inicialización escrito en C. A continuación lea el código fuente de *pintos_init()*.
No es necesario que remita nada relacionado con estas últimas indicaciones.

2.4. Ejercicio 3

Mejore threads/init.c para implementar un pequeño monitor de kernel en Pintos. Para ello, haga uso de la línea con el comentario

```
// TODO: no command line passed to kernel. Run interactively
```

Puede añadir nuevos ficheros fuentes en el código Pintos, por ejemplo, proveer una función de biblioteca para leer una línea. Este micro minomitor deberá ser capaz de:

[7] "Breakpoint"
[8] "Boot sector"

- Escribir en la terminal un "prompt" que se lea

  ```
  Shell ISC364>
  ```

- Leer el comando tecleado hasta el final de la línea. Se supondrá que el comando concluye con *Enter*.
- Si lo tecleado es *whoami*, debe escribir su nombre y presentar nuevamente el "prompt".
- Si lo tecleado es *exit*, debe concluir el monitor y permitir que pintos concluya.
- Para cualquier otra cosa tecleada, se escribirá "Comando desconocido" y presentar nuevamente el "prompt".

Sugerencia: Puede ser que necesite usar algunas funciones provistas en lib/kernel/console.c, lib/stdio.c y devices/input.c.

3. Productos

1. Documento pdf de nombre *PDE2.pdf* precedido por su matrícula sin guiones que contenga:
   - Título de la asignación, su nombre y su matrícula.
   - Las preguntas correspondientes al ejercicio 1 y sus respuestas. Este ejercicio recibe el 30 % de la nota máxima.
   - Las preguntas correspondientes al ejercicio 2 y sus respuestas. Este ejercicio recibe el 30 % de la nota máxima.
   - Ubicación de cada uno de los ficheros desarrollados y/o modificados dentro del proyecto pintos como respuesta del ejercicio 3.
2. Ficheros desarrollados y/o modificados para el ejercicio 3. Cada uno de los ficheros desarrollados y/o modificados deben estar comentados adecuadamente según los convenios de *Doxygen*.
3. Capturas de pantalla correspondientes a la ejecución de los comandos *whoami*, *exit* y alguno diferente en el ejercicio 3.

El ejercicio 3 recibe el 40 % de la nota máxima.

Todos los productos anteriores deben ser remitidos a la cuenta de correo del profesor en la PUCMM desde su cuenta de correo de estudiante. El nombre del fichero empaquetado *tar.xz* tendrá por nombre *<matrícula sin guión>carga.tar.xz* donde debe reemplazar *<matrícula sin guión>* por su matrícula sin guión intermedio.
Por ejemplo, si su matrícula fuese *2021-0777*, el fichero debería llamarse *20210777PDE1.tar.xz* El asunto del correo, siguiendo con la misma matrícula como ejemplo, sería *[20210777] ISC-364 PDE2. Análisis de carga.*