Universidad de San Andrés

I301 Arquitectura de computadoras y Sistemas Operativos

TP-5 File System Unix v6

Profesor: Daniel Veiga

Ayudantes: Matias Lavista, Nicolas Romero

Entregado:

Entrega: 08/06/24 a las 11:59 p.m.

Entrega tarde (máxima nota 8): 10/06/24 a las 11:59 p.m.

Importante: La resolución del TP es de manera individual

Trabajo Práctico basado en un TP de Stanford de la Materia “Principles of Computer System”

0. Set up:

Para poder compilar y correr el TP necesitan instalar lo siguiente:

sudo apt-get install libssl-dev
sudo apt-get install valgrind

1. Introducción:

Su trabajo consiste en escribir un programa que entienda el sistema de archivos Unix V6 para extraer datos de este. Para obtener una copia del código inicial, se debe clonar el siguiente repositorio:

tp5-acso_git

https://github.com/matiaslavista/tp5-acso

El proyecto se puede construir ejecutando el siguiente comando:

$ make

este generará un ejecutable **diskimageaccess** que se puede probar de la siguiente manera:

$ ./diskimageaccess <options> <diskimagePath>

que se puede comparar su salida contra la de la cátedra:

$ ./samples/diskimageaccess_soln <options> <diskimagePath>

diskimagePath: debe ser la ruta a uno de los discos de prueba ubicados en:

/samples/testdisks.

en testdisks, hay tres discos de prueba: *basicDiskImage, depthFileDiskImage y dirFnameSizeDiskImage.*

El ejecutable **diskimageaccess** reconoce solo dos banderas como opciones válidas:

- i: prueba las capas de inode y archivo.
- p: prueba las capas de nombre de archivo y ruta.

Por ejemplo, para ejecutar ambas pruebas de inode y nombre de archivo en el disco **basicDiskImage**, se puede ejecutar:

$ ./diskimageaccess -ip /samples/testdisks/basicDiskImage

La salida esperada de ejecutar diskimageaccess en cada imagen de disco X se almacena en el archivo X.gold dentro del directorio testdisks.

2. Esquema del código

Los archivos que les proporcionamos se dividen en tres categorías:

1. Los archivos de encabezado de Unix versión 6: **filsys.h, ino.h, direntv6.h**

   En el código inicial proporcionamos estructuras C correspondientes a las estructuras de datos en disco del sistema de archivos. Estas estructuras tienen el mismo diseño en la memoria que las estructuras en el disco.

   Incluyen:

   estructura filsys (filsys.h)

   Corresponde al *superbloque* del sistema de archivos. Esta es una copia ligeramente modificada del archivo de encabezado de la Unix V6.

   inodo de estructura (ino.h)

   Corresponde a un *inodo*. Nuevamente, esto proviene de la versión 6 de Unix, con algunas pequeñas modificaciones.

   estructura direntv6 (direntv6.h)

   Corresponde a una entrada del directorio.

2. Los archivos de prueba: **diskimageaccess.c, chksumfile.[ch], unixfilesystems.[ch]**

Estos archivos proporcionan la infraestructura para construir el código y ejecutar nuestras pruebas en él. **unixfilesystem.h** contiene una descripción del diseño del sistema de archivos, incluida la dirección del sector del superbloque y del inicio de la región del inodo.

3. Módulo del sistema de archivos: **diskimg.[ch], inode.[ch], file.[ch], pathname.[ch]**

   Estos son los archivos que van a tener que programar. El código de prueba que le brindamos interactúa con el sistema de archivos en capas y para cada una de estas se necesitará de su parte una implementación.

   Hay un archivo de encabezado (.h) que define la interfaz de la capa y un archivo (.c) correspondiente que debe contener la implementación.

Las capas son:

- Capa de bloque (**diskimg.[ch]**)

  Esto define e implementa la interfaz para leer y escribir sectores (recuerden que para nuestro caso bloque y sector son lo mismo) en la imagen del disco. Le brindamos una implementación de esta capa que debería ser suficiente para todo lo que necesitan.

- Capa de inodo (**inode.[ch]**)

  Deben implementar la interfaz para leer los inodos del sistema de archivos. Esto incluye la capacidad de buscar inodos por número y obtener el número de bloque/sector del enésimo bloque de datos del inodo.

- Capa de archivo (**file.[ch]**)

  Deben implementar la interfaz para leer bloques de datos de un archivo especificando su número.

- Capa de nombre de archivo (**directory.[ch]**)

Deben implementar la interfaz para directorios sobre archivos. Su función principal es obtener información sobre una única entrada del directorio.

- Capa de nombre de ruta (**pathname.[ch]**)

  Deben implementar la interfaz para buscar un archivo por su nombre de ruta absoluto.

3. Sugerencias e implementación

Les sugerimos resolver el TP en el siguiente orden:

- **inode_iget** e **inode_indexlookup** en inode.c.
- **file_getblock** en file.c.

Después de este paso, su código debería pasar las pruebas de funcionalidad a nivel de inodo. Luego resuelvan:

- **directory_findname** en directory.c.
- **pathname_lookup** en pathname.c.

Todas las pruebas de nombre de ruta deberían pasar.

Recordatorio:

+ Los bloques de large file son 7 indirectos y uno doblemente indirecto
+ Los bloques de un file que no es large son todos directos a data.
+ el primer inode es el inode 1
+ No tienen que manejar el caso de // en **pathname_lookup**
+ Inodes arrancan en `INODE_START_SECTOR`
+ **file_getblock** retorna la cantidad de bytes válidos del bloque que está buscando.

Extras:

i_mode del inodo:

El entero de 16 bits **i_mode** en la estructura del inodo no es realmente un número; más bien, los bits individuales del campo indican varias propiedades del inodo. ino.h contiene #defines que describen lo que significa cada bit.

Por ejemplo, decimos que se asigna un inodo si apunta a un archivo existente. El bit más significativo (es decir, el bit 15) de i_mode indica si el inodo está asignado o no. Entonces, la

expresión C (i_mode & IALLOC) == 0 es verdadera si el inodo no está asignado y falsa en caso contrario.

De manera similar, el bit 12 de i_mode indica si el archivo utiliza el esquema de mapeo de archivos grandes. Entonces, si (i_mode & ILARG)**!=** 0, entonces los campos i_addr del inodo apuntan a bloques indirectos y doblemente indirectos en lugar de directamente a los bloques de datos.

Los bits 14 y 13 forman un campo de 2 bits de ancho que especifica el tipo de archivo. Este campo es 0 para archivos normales y 2 (es decir, 10 binario o la constante IFDIR) para directorios. Entonces la expresión (i_mode & IFMT) == IFDIR es verdadera si el inodo es un directorio y falsa en caso contrario.

El primer inodo:

Como no hay ningún inodo con un número igual a 0, los diseñadores del sistema de archivos decidieron no desperdiciar los 32 bytes de espacio en disco para almacenarlo. El primer inodo del primer bloque de inodos tiene el **número 1**; este inodo corresponde al directorio raíz del sistema de archivos. (unixfilesystem.h para obtener más detalles).

3. Testing

Pueden modificar cualquiera de los archivos que le proporcionamos pero cuando se realicen cambios háganlo de manera compatible con versiones anteriores para que los scripts de prueba que le brindamos sigan funcionando. En otras palabras, está bien agregar código de prueba, pero asegúrese de que cuando ejecutemos su programa con -i y -p, su salida tenga el mismo formato con o sin sus cambios, para que nuestras pruebas automatizadas no se rompan.

Evaluamos:

+ Que no haya fuga de memoria.
+ Que manejen malos inputs de los usuarios (inodo inexistente, bloque inexistente, etc …). **Muy importante** en código de muchísimo uso.
+ Eficiencia del código.
+ Código ejecuta correctamente.