CSE451 Winter 2012 Project #4
Out: February 8, 2012
Due: March 5, 2011 by 11:59 PM (late assignments will lose ½ grade point per day)

Objectives

This project introduces you to the Windows FAT file system. FAT is the original disk format used by DOS, and is still used for floppy drives and flash media. You will be modifying a fully functional version of FAT. The principal objectives of this project are:

1. To gain knowledge of the functions and implementation strategies of a file system
2. Delve into one of the most complex parts of the Windows operating system

Getting Started

The build environment that we are going to use for FAT is not the same as for the first two projects, and more closely resembles the build environment used by the original Windows NT developers.

A complete copy of the source code is located in o:\cse\courses\cse451\12wi\Project4. Here are the step-by-step instructions to help you get started.

1. Make a private copy of the sources

Copy the entire project4 directory onto your assigned project workspace. For example on my account I copied everything to z:\project4.

```
xcopy /DECHKY O:\cse\courses\cse451\12wi\Project4 z:\project4
```

2. Setup the build environment

Then from a cmd window run z:\project4\wrk.cmd. This will setup the path and aliases you need to build the kernel. The last line of the command script is very important because it runs a batch script called setenv.bat.

```
z:\project4\bin\setenv.bat z:\project4\ fre WXP
```

3. Build fastfat.sys

All of the drivers in Windows NT have a .SYS filename extension. It would make common sense for the FAT file system driver to be called FAT.SYS, but because of the way it evolved, the driver is called FASTFAT.SYS.

So in the wrk window you need to cd down into the fastfat directory.

```
cd \project4\filesys\fastfat
```

There is also an alias “fastfat” that will transport you instantly to the fastfat directory.

```
fastfat
```

Once you are in the fastfat directory you can build the driver by issuing a build command.

```
build
```

The build program scans and constructs the file dependency list for all modules and then invokes nmake to actually run the compiler. You can see all the build switches by typing “build -?” The actual source files for fastfat.sys are located in the wxp subdirectory.

// Another alias is the “bz” command which forgoes the dependency scan. This command is useful if only .c file has been altered. Also in this build environment you can generate assembly code file, by cd’ing into the wxp directory and issuing the command “nmake filename.cod” For example “nmake read.cod” creates the file read.cod which a readable listing of the source code interspersed with the generated assembly code. //

4. Copy fastfat.sys to your image

The object modules, driver image and symbol file can be found in the subdirectory wxp\objfre_wxp_x86\i386. You want to copy your version of fastfat.sys into the drivers directory of your vhd test disk. The driver directory is \windows\system32\drivers. I suggest starting with a fresh differencing disk as you did in project 1. Also make a backup copy of the original fastfat.sys. For example if the vhd is c: and z: is the source the commands are:

```
c:
cd \windows\system32\drivers
ren fastfat.sys fastfat.sys.original
z:
cd z:\project4\src\filesys\fastfat\wxp\objfre_wxp_x86\i386
copy z:fastfat.sys c:fastfat.sys
```

Unlike the Windows kernel which lets you specify in the boot.ini file which kernel to boot, fastfat.sys is “hardwired” into the system. So if your system no longer boots because of changes you’ve done to fastfat.sys you will have to start over again with a new test disk.

Boot with the proper image and source path in windbg

While not necessary, it will probably be convenient to have debug source and symbols for the kernel available to you. To set this up I would rebuild the original project1 kernel.

Then after starting winbag be sure to expand the source, symbol, and image paths to include the kernel and fastfat. For example, my image and symbol path is:

```
z:\project1\base\ntos\build\exe;z:\project4\src\filesys\fastfat\wxp\
objfre_wxp_x86\i386
```

And my source path is:

```
z:\project1\base\ntos;z:\project4\src\filesys\fastfat\wxp;
```

5. Attach the FAT vhd and mount the vfd

The TestScripts directory contains copies of an empty floppy disk and two empty hard disks, all formatted as FAT. The format of FAT varies slightly based on the size of the disk; that is why there are three disks, one of each variety. Hard disks are attached to the system via the Virtual PC console before booting the test system. I suggest attaching both the 20MB and the 128MB to Hard Disk 2 and 3 respectively. Floppies on the other hand need to be captured after the system is booted.

Your assignment:

Before reading any further you might want to learn the format of a FAT volume. The following web page contains a description of the FAT file system's disk format, http://technet.microsoft.com/en-us/library/cc750198.aspx, also the Wikipedia page is pretty good: http://en.wikipedia.org/wiki/FAT_file_system.

Each directory in FAT is made up of a packed collection of directory entries (called dirents). A dirent is exactly 32 bytes in size. Originally FAT only supported 8.3 names (aka short names) and there was exactly one dirent per file (see fat.h for a complete definition of a dirent). Later this was extended to support long names (i.e., file names that are greater than 8.3), but for this project you can limit yourself to short names.

The dirents in FAT are not inserted in any sorted order. Every time a new file is created the file system assigns to it the first available dirent. And when a file is deleted the dirent is marked as deleted and the file system is free to recycle it.

Your assignment is to keep the directories on disk sorted and compacted. Directory entries can be sorted by filename, extension, size, time, or initial fat cluster. You will decide how and if a volume is sorted based on its volume label. Possible labels are "SORTBYNAME", "SORTBYEXT", "SORTBYSIZE", "SORTBYTIME", and "SORTBYFAT". A volume not containing one of these labels is not sorted or compacted.

Note that volume labels can change at any time and the behavior of the file system must track these changes.

The TestScript directory contains a few batch scripts to help test your code. The main scripts are testname.cmd testext.cmd, testsize.cmd, testtime.cmd, and testfat.cmd which should exercise each basic sorting option. In addition there is testrelabel.cmd which

cycles through relabeling the volume. They each take an optional drive letter as their first parameter, which if not specified defaults to "a:"

Caveats:

Keeping the root directory sorted is your primary objective. However you will probably find it easier and more natural to expand your scope to include sorting and compacting subdirectories. You will also discover as you look through the sources that the root directory on smaller disks that are not formatted as FAT32 are special, because they are in a fixed area on the disk and not allowed to grow or shrink. You should handle both situations.

You will also be required to correctly handle the situation where the disk is write-protected. So work through the write-protect scenario to make sure you understand all of the ramifications.

Limitations (aka extra things to do to get an "A"):

What should you do if you encounter an existing directory that is not sorted? If you do encounter such a directory you can have the system fall back to its original behavior of not having sorted dirents; however, it would be really nice if you did sort these preexisting directories before deleting or creating new files. But only sort the directory if you are modifying files in the directory, and not just reading files.

As stated earlier, you do not need to support directories with long names, and if you do encounter long names you can revert back to the original behavior. However for extra credit you can add long name support.

Windows NT File System Architecture

The main interface between the Window I/O system and the file system is through a set of predefined entry points (called FSD Entry points). Each FSD entry point is responsible for a major I/O function. They take as input a pointer to an object representing the volume being accessed and a structure called an IRP (I/O Request Packet). IRPs are defined in \project4\inc\ddk\wdm.h. Essentially the IRP provides the file system everything it needs to know in order to perform the requested action. Here is a list of the FSD entry points and the file where they are defined.

- **create.c – FatFsdCreate** (Create and open files and/or directories)
- **close.c – FatFsdClose** (Called when the OS is entirely finished using the file)
- **read.c – FatFsdRead** (Read data from a file)
- **write.c – FatFsdWrite** (Write data to a file)
- **fileinfo.c – FatFsdQueryInformation, FatFsdSetInformation** (Query/set file information)
- **ea.c – FatFsdQueryEa, FatFsdSetEa** (Query/set Extended attribute information, now obsolete)

- **flush.c – FatFsdFlushBuffers** (Flush whatever the file system has cached)
- **volinfo.c – FatFsdQueryVolumeInformation, FatFsdSetVolumeInformation** (Query/set information about the volume)
- **cleanup.c – FatFsdCleanup** (Called when a user handle is closed)
- **dirctrl.c – FatFsdDirectoryControl** (Query the contents of a directory)
- **fsctrl.c – FatFsdFileSystemControl** (Various control/maintenance functions)
- **lockCtrl.c – FatFsdLockControl** (Does byte range locking)
- **devCtrl.c – FatFsdDeviceControl** (Used to pass control operations down to a lower level driver)
- **shutdown.c – FatFsdShutdown** (Called when the system is going through a shutdown)
- **pnp.c – FatFsdPnp** (Used for PNP support)

The main header files you need to look at are located in fastfat\wxp and in \project4\inc\ddk directory. Within fastfat the header files are:

- **fat.h** – defines the on-disk structure of a fat volume
- **fatdata.h** – defines the global data used by fastfat
- **fatprocs.h** – defines all the routines that can be called between various fastfat sources modules.
- **fatstruc.h** – defines the major internal data structures used by fastfat to represent the disk volume, opened directories, opened files, etc.
- **lfn.h** – defines the structures of long file names on FAT.
- **nodetype.h** – defines some of the constants used to develop and debug the system

Here is the reminder of the source files making up fastfat.

- **acchksup.c** – Implements routines that support doing access checks.
- **allocsup.c** – Implements routines that support disk space allocation.
- **cachesup.c** – Support routines that help fastfat interface with the cache manager.
- **deviosup.c** – Support routines for doing low level read/write operations to the disk.
- **dirsup.c** – Support routines for managing dirents.
- **dumpsup.c** – Debug routines that were useful when developing the file system
- **easup.c** – Support routines for manipulating extended attributes on FAT.
- **fatdata.c** – Declares the structures defined in fatdata.h.
- **fatinit.c** – Contains the routine called by the I/O system when loading fastfat.sys.
- **filobsup.c** – Used to store/retrieve file system specific information that it stores in a file object.
- **fspdisp.c** – A main dispatch routine used if a secondary thread (i.e., a worker thread) is processing an item off a work queue.
- **lockctrl.c** – Support routines to help manage file locks.
- **namesup.c** – Support routines to help manage file names (both short and long file names).
- **resrcsup.c** – Support routines that manage synchronizing access to our in-memory data structures.
- **splaysup.c** – Support routine to help search for already opened file names.

- **strucsup.c** – Support routines to manage our in-memory data structures.
- **timesup.c** – Support routines to manage converting between NT time and the format stored on disk.
- **verfysup.c** – Support routines to help us verify a volume that has been removed and then reappears.
- **workque.c** – Handles posting work items for later processing.

For this project you will probably want to start by examining create.c. In particular look at the logic for the function FatCommonCreate and then branch out for there.

Turn-in:

Be prepared to turn in the following

1. Your compiled and linked version of fastfat.sys
2. Source code for your changes
3. A write up listing the two students who worked on the project and summarizing changes.

You'll be submitting the source code, executables, and write-up to Catalyst.

Grading Criteria:

Your project will be graded based on the following criteria:

E (0.0) - For turning in nothing and/or doing nothing
D (1.0) - For a project that is dysfunctional. It may not compile or it gives completely bogus answers.
C (2.0) - For a project that is barely functional. Or it might miss in a few cases but generally it is okay. Or you can sort of understand how the code works but it is not clear.
B (3.0) - For a project that is functionally complete. It is clean and does the job but without any real elegance. It is a meat and potatoes type solution. So a "B" simply accomplishes the objectives, but nothing fancy.
A (4.0) - Is for projects that go well beyond just being functional. They are works of art. We can look at an "A" project and it would stand high above the "B" project.

There is nothing wrong with a "B". It demonstrates clear mastery of the subject. Late assignments will lose ½ a grade point per day.