Honors Project Overview

Stewart Smith,
Monash University, Clayton, Australia
email: sesmith@csse.monash.edu.au

April 28, 2003

Abstract

"Walnut Kernel": To design and implement a data storage system usable on Walnut and prototyped on UNIX with advanced features such as user visible transactions and versioning.

1 Introduction

The current on-disk format for the Walnut Kernel lacks certain features and scalability needed if it is to enter wider use. Current UNIX filesystems also lack some advanced features that could be useful to both users and system administrators. Since a lot of work has been done to implement a UNIX like environment on top of Walnut, and the use of a nameserver to translate paths to capabilities is expected to be common it should not be too hard to design a system that would be compatible with both systems. Plus, it would just be plain cool.

2 Object Store

Store objects on disk. Referenced by an object ID. Lookup of an object by Object-id should be very quick but should also be scalable should the disk size increase (i.e. by partition enlarging).

3 Requirements

3.1 Meta-data Consistency

If we lose our metadata, we've got no real hope of reliably getting our data, so this is pretty important. Although, performance should always be considered.

3.2 Data Consistency

Even with keeping metadata consistency it is possible to have the contents of a file non-consistent. This could happen when a process is modifying a file (this operation doesn't require filesystem meta-data update) and we experience

a power failure half way through this update. On disk, half will be old data and half will be new data. Eliminating this could stop a lot of data file corruption (especially for binary file formats, which are difficult to recover).

3.3 Inter-object consistency

Links between different stored objects (e.g. a mail file and an index file) could be inconsistent if the copy on disk of one is more recent than the other. This is an age old problem that has been solved by the database people via the use of transactions.

3.4 Object Size Support

Some research into patterns of usage may be required. As a guess, files are getting larger (on average) and there are more large files than there were several years ago (what with the advent of MP3, Ogg and DivX). Good support for large (and VERY large) files should exist as well as non-sucky support for very small files (¡4k). Direct, indirect, double indirect (and maybe 3rd indirect) block pointers could be useful.

3.5 Speed of Object Lookup

Object lookup time should be as close to O(1) as possible.

3.6 Read&Write Speed

Once an object is looked up and opened, read&write data rates should be as close to raw block device as possible. Being faster is an option, but usually against the laws of physics.

4 Tests

5 Object Forks

The MacOS has traditionally had two parts to a file: the resource for and the data fork. This can be useful in seperating data from meta-data (many Mac only word processors stored the text as text in the data fork, and all the styling and formatting information in the resource fork).

Having several 'forks' to an object could be useful in

- Supporting use on Darwin/MacOS X as a primary drive
- adding versioning support to the object store. We could have a 'versioning' fork where all the diffs are stored.
- storing Walnut data structures (i.e. information like: capabilities, money, last rent collected)

6 Versioning

- RCS/CVS
- WAFL[1]: Takes Snapshots of the FS by duplicating the root inode and doing copy-on-write.
- Episode: fileset clones
- xdfs

7 Consistency

Linux ext2 loves to ignore everything for the sake of speed. fsck can fix it all.

WAFL[1] creates a snapshot every few seconds[1] to allow quick recovery after an unclean shutdown. About a minute for 20GB data[2]. Also keeps a log structure of NFS[3] writes in NVRAM[4] that it replays after an unclean shutdown.

8 Object Size and usage trends

To get an efficient system, we'll need to look at how people (and systems) use an object store. If we take the Walnut idea of having an Object pretty much equivilent in function to a file, then we should be able to look at file usage on UNIX to get an idea on object usage.

However, UNIX is modelled around the concept of a file, so our statistics may be skewed. It is perhaps also a good idea to look at non-unix systems such as the Classic MacOS[5], BeOS (which although has aimed towards POSIX compliance, it fundamentally isn't a UNIX) and Windows NT[6]

9 Object Lookup

To locate an object on disk, an appropriate object ID must be provided. This ID must be looked up in some kind of on-disk catalog to find the physical location of the object as the whole of the object cannot be stored in the catalog.

10 Policy

We can do a lot of things with how Walnut or UNIX behaves in regard to the on-disk structure simply by specifying a policy.

[1] every 10 seconds so the FS is not too out of date

[2] This is a network appliance system, so it's probably CPU limited

[3] Network File System

[4] Lyon and Sandberg describe NVRAM write cache technique. Legato's Prestoserver NFS accelerator usess this[1]

[5] MacOS X is UNIX based

[6] The current incarnations are: ME, 2000 and XP.

11 Useful Resources

I've already read into a fair bit on file system design, and list some of these references below:

- File Structures[2]
- Practical File System Design with the Be File System[3]

12 Evidence of achievability

Two engineers at Be implemented a new VFS (including device interface) and the on-disk BeFS in 9 months (it shipped non-beta the next month)[3]. Since I already have access to a mature VFS in Linux, that eliminates a fair amount of work (maybe even half). If we remove the desire to have a fully functional, release quality product (or even Beta quality) within the project timeframe, then the project starts to sound a lot more feasable within the desired timeframe (i.e. before the thesis is due)

Ron has said that it's probably better to focus on getting the ideas down instead of simply an implementation.

References

[1] James Lau Dave Hitz and Michael Malcolm. File system design for an nfs file server.

[2] Michael J. Folk and Bill Zoellick. *File Structures: A Conceptual Toolkit*. Addison-Wesley Publishing Company, 1987.

[3] Dominic Giampaolo. *Practical FileSystem Design with the Be File System*, chapter 1. Morgan Kaufmann Publishers, Inc., 1999.

[4] Squid Team.