Research Proposal

Revision: 1.10

Stewart Smith
sesmith@csse.monash.edu.au

Contents

Abstract

"Walnut Kernel": To design and implement a data object storage system with advanced features such as user visible transactions and support for object versioning.

1 Introduction

1.1 Walnut

Walnut[4] is a persistent password capability based operating system extending many of the ideas first presented in the Password Capability System[17]. Being a persistent system, all state information is saved on shutdown and when you restart the system, it should bring you back to **exactly** the way it was before shutdown. During recent work on the Walnut Kernel, it has become apparent that areas of the data storage system would be inadequate for use in a production system where data consistency and reliability are priorities.

1.2 Walnut Storage

Processes running on Walnut access blocks of data (objects) referenced by a unique object ID. These objects are then mapped into the address space of the process. Conceptually, this is similar to the UNIX mmap function. A user running Walnut would rely on a process called a nameserver to do translation from a human readable format (such as a pathname) to a native object ID. It is thought that a UNIX like system could be implemented on top of Walnut and the nameserver would be the equivalent of the part of the UNIX kernel which maps pathnames to inodes.

As Walnut is essentially just a large virtual memory system, swift lookup and access to objects (or parts of objects) is very important for system performance. Some systems, such as the Berkeley Fast File System have traditionally relied on synchronous writes to ensure consistency on disk; this is a solution that is not possible on a system such as Walnut due to the performance impact of effectively reducing main memory speed to that of disk.

Currently, Walnut objects must be a multiple of the page size of the architecture it runs on (currently only x86). There is a strong desire to remove this limitation and the data storage system should encourage this.

1.3 Walnut Persistence and consistency

As Walnut is a persistent system, on-disk data consistency is of greater importance to it than many other systems. On a UNIX system, if the contents of a file is corrupted after a crash, the process requiring that file may refuse to start or show a warning message. On Walnut, since processes are preserved across reboots[1] if a data object were to become corrupted, a running process could have data it was using suddenly change underneath it, an error condition that would be nearly impossible to detect.

There is also the issue of inter-object dependency. If object A is a list, and object B is an index of this list, then they should be kept in sync with each other. In a system such as Walnut, if these were to be out of sync on disk when an unclean shutdown took place the processes (when resumed) would be dealing with corrupted data and possibly leave it in an unrecoverable state. This situation should be avoided, even if extra work is needed as users do not like mysterious failures.

The database community has long enjoyed isolated transactions to ensure inter-object consistency but transactions in filesystems have usually fallen short of this[18]. Having user visible transactions as a way to ensure consistency of the contents of objects could greatly increase the reliability of a persistent system such as Walnut.

1.4 Object Revisions and Versioning

Another feature that programmers have long enjoyed with source code management systems such as RCS, CVS, Katie and BitKeeper is versioning. Being able to quickly have a look at what a file was like yesterday, or files that have been deleted is considered extremely useful by programmers. Many system administrators (and users) have wished that they could retrieve the way a file was 10 minutes ago (or even yesterday). Tracking object versions could be something that people find invaluable, and not have too

[1]The init process being the one exception

much computational or storage overhead[19]. With a system such as Walnut, it would then be possible to 'replay' certain actions in the system if the need arose.

1.5 Aims

I aim to design a data storage system for the Walnut Kernel that has the following properties:

- Fast
- Efficient
- Guarantees consistency
- support for tracking of object revisions

Due to time constraints (and the possible complexity involved) it may not be possible to implement full object revision support, but I aim to design the system with this in mind so it is possible to implement at a later date.

I will implement the prototype system on top of the Linux Kernel as it is more mature than the currently Walnut source and runs on a larger variety of hardware. Because of this, issues with integrating these features with a traditional UNIX like environment will also be explored.

2 Research Context

2.1 Consistency in Persistent Systems

In research into persistent operating systems, both single address space[13] and object based[4] systems have generally glossed over how persistence is provided in the event of an unclean shutdown (e.g. a system crash or power failure). The on-disk structures are detailed and sometimes provide a rather efficient storage system but what is not clear is how these structures (and the data on disk) is kept in a consistent state.

These systems allow processes to access and modify data very easily as all data is memory mapped. Walnut[4] periodically flushes mapped objects to disk as its way of providing persistence. Walnut makes no guarantee about when objects will be written to disk or that the process of writing all active objects to disk will complete successfully. In the event of a crash, Walnut relies on an as-yet unwritten fsck[2] like utility to maintain meta-data consistency and it completely ignores the possibility of data inconsistency.

2.2 Comparison with UNIX Semantics

This is different from the more conventional UNIX like semantics of open(), read(), write() and close() where the assumption can be made that an object (file) will be self consistent after the close() system call. UNIX still ignores any ideas of inter-object consistency (file A relying on file B). Efforts have been made[26][27][3] to improve the reliability of UNIX filesystems and help avoid the reliance on an fsck utility for meta-data consistency. A common method has been to adopt a journaling strategy[24][18] where meta-data transactions are logged and are either completed fully or not at all. Many database systems[3] employ a similar technique but with the addition of having isolated transactions, which allow them to have a rollback capability. That is, the ability to abort an individual transaction programmatically (that is, not as the result of a crash). ReiserFS[18] has indicated that this could be supported in the future but currently, the transaction interface is only exported to trusted modules.

2.3 Filesystem Metadata Consistency

The Berkeley Fast Filesystem has traditionally relied on synchronous meta-data updates to help ensure a certain level of metadata consistency. The FreeBSD project has improved on the reliability and performance of their implementation of the Berkeley Fast Filesystem by adding Soft Updates[21]. Soft updates aims to ensure consistency by carefully tracking and enforcing metadata update dependencies so that the on disk image is always consistent. The only points of inconsistency with soft updates are unclaimed inodes or blocks, both of which can be fixed easily with a (now backgrounded) fsck[4]. The free BSDs

[2]File System ChecK, a program (fsck on UNIX) that checks a file system for errors in the on-disk data structures and optionally attempts to repair them. This is typically run after a non-clean shutdown

[3]PostgreSQL (http://www.postgresql.org) is one such system

[4]File System ChecK

(FreeBSD, OpenBSD, NetBSD) are the only known Operating Systems to currently implement a Soft Updates like system.

A common way that filesystems keep consistent metadata is by maintaining a log of operations on the filesystem which can be replayed to bring the filesystem metadata back to a consistent state after a crash. Write ahead logging (where the details of an operation are written to the log before being committed to disk) has been used by Database Management Systems for many years. More recently, filesystems have used these techniques to help maintain metadata consistency. Various variations on journalling can now be found in ReiserFS[26], Reiser4[18], HFS Plus[16], XFS[24], BeFS[12], NTFS, Network Appliances[14] and ext3.

2.4 Revision tracking

Both RCS[37] and XDFS[19] have shown that tracking deltas between files (usually different revisions of the same file) can be done efficiently. Time and space requirements are approaching minimal by todays standards. Since one of the most common wishes is to be able to see what a file was like yesterday (before you changed everything and broke it) or be able to get back that file you deleted last week (when you hit the wrong key) being able to track revisions on objects could be invaluable. Following the simple rule of disk space is cheaper than a person's time, a compelling case can be made for tracking every revision to a system. Several Log based file systems have had this ability, but few have provided an interface to it.

3 Research Plan and Methods

3.1 Research Methods

This project will build on existing research into filesystems and databases. Certain techniques will need to be extended and adapted to suit the target (and test) environments. It has become clear that several ways of doing things may have to be melded and compared before the aims can be fully met. Experiments will need to be conducted on the system to evaluate reliability and efficiency under real world conditions using various techniques. This data will be helpful in determining what techniques perform well in the real world, something which is important to people using a system.

3.2 Proposed Thesis Chapter Headings

1. Introduction
   (a) Purpose of Research
   (b) Objectives of Research
2. Storage In Walnut
   (a) Current System
   (b) Requirements for new system
3. Consistency
   (a) Metadata consistency
   (b) Inter-Object consistency
4. Versioning
   (a) Requirements For Versioning
   (b) Proposed Implementation details
5. Design
   (a) On-Disk Structures
   (b) Algorithms
   (c) Policy
6. Conclusion and Future Work

7. Bibliography
8. Appendix A: Source Code
9. Appendix B: Test Suite
10. Appendix C: Test Data

3.3 Timetable

There is no doubt that this timetable will be revised and that my estimates of dates will prove grossly inaccurate. I prefer to define success by 'satisfactory progress and effort' rather than meeting strict dates for milestones.

Date	Sem/Week No.	Activity
20-05-2003	1/11	Begin Literature Review
28-05-2003	1/12	Research Methods Assignment Due
28-05-2003	1/12	Begin Preparing for Interim Presentation
30-05-2003	1/12	Have draft of Specification completed
30-05-2003	1/12	Begin coding from draft specification
05-06-2003	1/13	Interim Presentation
11-06-2003	1/14	Literature Review Draft
10-07-2003	-	Have working example
30-07-2003	2/2	Literature Review Due
30-07-2003	2/2	Have a complete specification document
15-08-2003	-	Complete coding
10-09-2003	2/8	First Thesis Draft
11-11-2003	-	Web Site
04-11-2003	-	Research Log Book
04-11-2003	-	Final Report/Thesis

3.4 Facilities Needed

3.4.1 Testbed

The choice to develop on top of the Linux Kernel will require a machine capable of running the developmental series of the Linux Kernel (currently the 2.5 series). All development for the Linux Kernel is recommended to be done on 2.5 and then backported to 2.4 if needed. The most stable architecture for the development series is the x86. I will require the ability to install software and kernels on a testbed machine. It should **not** be used for anything else but the testing, just in case anything nasty happens as the result of bugs. This will have to be arranged by the university as it is not currently possible to install custom kernels on the honors lab machines

3.4.2 Development Machine

To compile the Kernel and my additions, a relatively fast x86 PC with decent disk speed and capacity. A reasonable amount of memory will also be required. A different setup from the honors lab machines will be needed as the software is grossly inadequate and my requests for the software have been ignored.

4 Relevance of the Proposal

Through this project, I aim to further the work already done with persistent Operating Systems and examine closely the issues of consistency across unclean reboots. This should increase the reliability of persistent systems and make way for their acceptance into markets that demand reliability. By implementing on top of UNIX, I aim to show that these ideas can also benefit more conventional operating systems in the same way, improving reliability.

References

[1] Curtis Anderson. xfs attribute manager design. Technical report, Silicon Graphics, 1993.

[2] Curtis Anderson. xfs namespace manager design. Technical report, Silicon Graphics, 1993.

[3] Curtis Anderson, Doug Doucette, Jeff Glover, Wei Hu, Michael Nishimoto, Geoff Peck, and Adam Sweeney. xfs project architecture. Technical report, Silicon Graphics, 1993.

[4] Maurice David Castro. *The Walnut Kernel: A Password-Capability Based Operating System*. PhD thesis, Monash University, 1996.

[5] Alan Dearle, John Rosenberg, Frans Henskens, Francis Vaughan, and Kevin Maciunas. An examination of operating system support for persistent object systems. In *Proceedings of the Twenty-Fifth Annual Hawaii International Conference on System Sciences*, pages 779–789, 1992.

[6] Doug Doucette. xfs kernel threads support. Technical report, Silicon Graphics, 1993.

[7] Doug Doucette. xfs message system design. Technical report, Silicon Graphics, 1993.

[8] Doug Doucette. xfs project description. Technical report, Silicon Graphics, 1993.

[9] Doug Doucette. xfs simulation environment. Technical report, Silicon Graphics, 1993.

[10] Doug Doucette. xfs space manager design. Technical report, Silicon Graphics, 1993.

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

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

[13] Gernot Heiser, Kevin Elphinstone, Jerry Vochteloo, Stephen Russell, and Jochen Liedtke. The Mungi single-address-space operating system. *Software Practice and Experience*, 28(9):901–928, 1998.

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

[15] Wei Hu. Uuids. Technical report, Silicon Graphics, 1993.

[16] Apple Computer Inc. Hfs plus volume format. Technical report, Apple Computer Inc.

[17] M. Anderson, R. D. Pose, and C S. Wallace. A Password-Capability system. *The Computer Journal*, 1986.

[18] Joshua MacDonald, Hans Reiser, and Alex Zarochentcev. Reiser4 transaction design document. Technical report, Namesys, 2002.

[19] Joshua P. MacDonald. File system support for delta compression. Technical report, University of California, Berkeley.

[20] Joshua P. MacDonald. File system support for delta compression. Technical report, University of California at Berkeley.

[21] Marshall Kirk McKusick and Gregory R. Ganger. Soft updates: A technique for eliminating most synchronous writes in the fast filesystem.

[22] Rodney Van Meter and Minxi Gao. Latency management in storage systems. Technical report, Quantum Corp. and U.C. Berkeley.

[23] Jim Mostex, Willam Earl, and Dean Koren. Porting the sgi xfs file system to linux. Technical report, Silicon Graphics.

[24] Michael Nishimoto. The log manager (xlm). Technical report, Silicon Graphics, 1994.

[25] Gultekin Ozsoyoglu and Richard T. Snodgrass. Temporal and real-time databases: A survey. *Knowledge and Data Engineering*, 7(4):513–532, 1995.

[26] Hans Reiser. Future vision. Technical report, NameSys, 2001.

[27] Hans Reiser. http://www.namesys.com/v4/v4.html. Technical report, Namesys, 2002-2003.

[28] Jiri Schindler, John Linwood Griffin, Christopher R. Lumb, and Gregory R. Ganger. Track-aligned extents: Matching access patterns to disk drive characteristics.

[29] Frank Schmuck and Roger Haskin. Gpfs: A shared-disk file system for large computing clusters. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*.

[30] Adam Sweeney. 64 bit file access. Technical report, Silicon Graphics, 1993.

[31] Adam Sweeney. xfs in-core inode management. Technical report, Silicon Graphics, 1993.

[32] Adam Sweeney. xfs superblock management. Technical report, Silicon Graphics, 1993.

[33] Adam Sweeney. xfs transaction mechanism. Technical report, Silicon Graphics, 1993.

[34] Adam Sweeney. xfs block zeroing mechanism. Technical report, Silicon Graphics, 1994.

[35] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system.

[36] Squid Team.

[37] Walter F. Tichy. RCS - a system for version control. *Software - Practice and Experience*, 15(7):637–654, 1985.

[38] C.S. Wallace and R.D. Pose. Charging in a secure environment. Technical report, Department of Computer Science, Monash University.

[39] Fusheng Wang and Carlo Zaniolo. Preserving and querying histories of xml-published relational databases. In *Proceedings of the Second International Workshop on Evolution and Change in Data Management (ECDM '02)*, Tampere, Finland, 11 October 2002.

[40] Chi Zhang, Xiang Yu, Arvind Krishmacmurthy, and Randolph Y. Wang. Configuring and scheduling an eager-writing disk array for a transaction processing workload.