$2.50

MUUGlines

October 1997 Volume 10, Number 2

This Month's Meeting

Arne Grimstrup has kindly arranged to have a representative from MTS provide us with a technical overview of the newest innovation in high speed residential data communications, ADSL (Asymmetric Digital Subscriber Line).

This technology promises up to megabits per second of data transferred right to your home PC, far surpassing today's not-very-interoperable 56Kbps modems. The presentation will include a description of how ADSL works; a comparison to ISDN; what happens at the subscriber and the exchange end; and what's being offered by MTS. Naturally, with this still being a product in development, the price and availability of the service will be guessed at wildly by the audience with no assistance from the MTS representative.

This month we'll have (yes!) some door prizes! We have about ten copies of OpenDOS from Caldera, and roughly 15 of OpenLinux Lite from the same fine company. We also have the monthly issue of the Linux Journal to give away and possibly others, so don't miss your chance to enter the draw. It's free with your attendance! Some prizes are restricted to members only, so feel free to sign up as you arrive.

Our meeting this month is Tuesday, the 14th of October. We'll be meeting at IBM Canada's offices in the TD Centre, at the corner of Portage and Main. We'll gather in the lobby on the main floor – please try to be there by about 7:15 PM. Steve Moffat will then take us up to the meeting room just before the meeting starts at 7:30. Don't be late, or you may not get in.

Parking is available either in the parkade behind the TD building, off Albert Street, or in the ground level lot just north of the TD building. Entrance to the lot is from Albert Street, behind the parkade. Either way, parking is a $1.25 flat rate for the evening. You purchase your ticket from a dispenser, so make sure you've got exact change – a loonie and a quarter, or 5 quarters.

Next Meeting

The Manitoba Unix User Group will next meet on November 18, 1997. That's right, due to the regular meeting date (the second Tuesday of the month) falling on November 11th (Remembrance Day), we have rescheduled the meeting for the third Tuesday of the month for November only. Check your mailbox and the web site for details on the upcoming meeting's program.

MUUG Display at AGM

Contributed by Doug Shewfelt

The Muddy Waters Community Networks group on Saturday, Sept. 27 held its inaugural Annual General Meeting at the St. Boniface Hospital Research Centre.

Muddy Waters Community Networks is a new organization set up to operate the Winnipeg node of the Blue Sky Freenet. The Blue Sky Freenet group began several years ago to build community networks in Manitoba, and their first pilot freenet was in Winnipeg. Since then, several other groups have started up freenets in Manitoba, under the umbrella of the Blue Sky Freenet.

Blue Sky Freenet decided that it should concentrate on developing policy and providing assistance to Manitoba groups operating freenets. It therefore wanted another group to take the responsibility of running the Winnipeg node on a day-to-day basis. Some members of the Muddy Waters Computer Society volunteered to help set up the MWCN group to run the node.

The Annual General Meeting went well, although organizers were visibly disappointed that the freenet membership offered only two nominations to the board of MWCN.

By invitation, MUUG had a display table set up in the atrium, along with The Winnipeg Apple User Group and Blue Sky Freenet.

We were able to show our web site and new mirror site for Linux software, as well as a desktop computer running a recent release of Red Hat Linux. We also had a free draw for four copies of Caldera's OpenLinux Lite. Although the turnout was small, there was a lot of interest in Linux, and we were impressed that the attendees asked some very good questions.

Special thanks go to Paul Hope for providing us with an Ethernet connection at the Research Centre.

FTP to MUUG!

If you're a frequent downloader of RedHat, GNU, Slackware, or MkLinux software, patches and distributions, there's a great service available to you!

MUUG is now mirroring several great sites. This means much less contention when you need those patches – no more 'too many users' messages from the FTP server. And no more slow downloads from those popular locations in congested Internet regions. If you're a MBnet member, you'll see fairly high efficiency transfers!

Details about how the site was set up was provided in last month's newsletter, in an article by Gilbert Detillieux, the site's maintainer. You can find that issue in several formats at our web site, www.muug.mb.ca. This month I want to remind you that Gilbert is still looking for suggestions about what else should be mirrored. If there are some parts of our mirror that are of less value to our members than something you have in mind, let us know! Send a message to the board (board@muug.mb.ca) or our webmaster.

Linux Gazette Excerpts

A Non-Technical Look Inside the EXT2 File System

By Randy Appleton, randy@euclid.nmu.edu

Introduction

Everyone wants a fast computer. However, not everyone realizes that one of the most important factors of computer performance is the speed of the file system. Regardless of how fast your CPU is, if the file system is slow then the whole computer will also seem slow. Many people with very fast Pentium Pros but slow disk drives and slower networked file systems rediscover this fact daily.

Luckily, Linux has a very fast file system called the Extended File System Version 2 (EXT2). The EXT2 file system was created by Remy Card (card@masi.ibp.fr). This article will show you how the EXT2 file system is organized on disk and how it gets its speed.

Disk Layout: Goals

There are several objectives when deciding how to lay data out upon a disk.

First and foremost, the data structure should be recoverable. This means that if there is some error while writing data to the disk (like a silly user pulling the power cord) the entire file system is not lost. Although losing the data currently being written is often acceptable, losing all the data on the disk is not.

Secondly, the data structure must allow for an efficient implementation of all needed operations. The hardest operation to implement is normally the hard link. When using a hard link, there are more than one directory entry (more than one file name) that points to the same file data. Accessing the data by any of the valid file names should produce the same data.

Another hard operation involves deleting an open file. If some application has a file open for access, and a user deletes the file, the application should still be able to access the file's data. The data can be cleared off the disk only when the last application closes the file. This behavior is quite unlike DOS/Windows, where deleting a file means that applications who have already begun to access the file lose all further access. Applications that use this UNIX behavior concerning deleted files are more common than one might think, and changing it would break many applications.

Thirdly, a disk layout should minimize seek times by clustering data on disk. A drive needs more time to read two pieces of data that are widely separated on the disk than the same sized pieces near each other. A good disk layout can minimize disk seek time (and maximize performance) by clustering related data close together. For example, parts of the same file should be close together on disk, and also near the directory containing the file's name.

Finally, the disk layout should conserve disk space. Conserving disk space was more important in the past, when hard drives were small and expensive. These days, conserving disk space is not so important. However, one should not waste disk space unnecessarily.

Partitions

Partitions are the first level of disk layout. Each disk must have one or more partitions. The operating system pretends each partition is a separate logical disk, even though they may share the same physical disk. The most common use of partitioning is allow more than one file system to exist on the same physical disk, each in its own partition. Each partition has its own device file in the /dev directory (e.g. /dev/hda1, /dev/hda2, etc.). Every EXT2 file system occupies one partition, and fills the whole partition.

Groups

The EXT2 file system is divided into groups, which are just sections of a partition. The division into groups is done when the file system is formatted, and cannot change without reformatting. Each group contains related data, and is the unit of clustering in the EXT2 file system. Each group contains a superblock, a group descriptor, a block bitmap, an inode bitmap, an inode table, and finally data blocks, all in that order.

Superblock

Some information about a file system belongs to the file system as a whole, and not to any particular file or group. This information includes the total number of blocks within the file system, the time it was last checked for errors, and so on. Such information is stored in the superblock.

The first superblock is the most important one, since that is the one read when the file system is mounted. The information in the superblock is so important that the file system cannot even be mounted without it. If there were to be a disk error while updating the superblock, the entire file system would be ruined. Therefore, a copy of the superblock is kept in each group. If the first superblock becomes corrupted, the redundant copies can be used to fix the error by using the command e2fsck.

Group Descriptors and Bitmaps
The next block of each group is the group descriptor. The group descriptor stores information on each group. Within each group descriptor is a pointer to the table of inodes (more on inodes in a moment) and allocation bitmaps for inodes and data blocks.

An allocation bitmap is simply a list of bits describing which blocks or inodes are in use. For example, data block number 123 is in use if bit number 123 in the data bitmap is set. Using the data and inode bitmaps, the file system can determine which blocks and inodes are in current use and which are available for future use.

Inodes and Such
Each file on disk is associated with exactly one inode. The inode stores important information about the file including the create and modify times, the permissions on the file, and the owner of the file. Also stored is the type of file (regular file, directory, device file like /dev/ttyS1, etc) and where the file is stored on disk.

The data in the file is not stored in the inode itself. Instead, the inode points to the location of the data on disk. There are fifteen pointers to data blocks within each inode. However, this does not mean that a file can only be fifteen blocks long. Instead, a file can be millions of blocks long, thanks to the indirect way that data pointers point to data.

The first thirteen pointers point directly to blocks containing file data. If the file is thirteen or fewer blocks long, then the file's data is pointed to directly by pointers within each inode, and can be accessed quickly. The fourteenth pointer is called the indirect pointer, and points to a block of pointers, each one of which points to data on the disk. The fifteenth pointer is called the doubly indirect pointer, and points at a block containing many pointers to blocks each of which points at data on the disk.

This scheme allows direct access to all the data of small files (files less than fourteen blocks long) and still allows for very large files with only a few extra accesses. As the table below shows, almost all files are actually quite small. Therefore, almost all files can be accessed quickly with this scheme.

File Size	Occurrence %	Cumulative %
0-768	38.3	38.3
769-1.5K	19.8	58.1
1.5K-3K	14.2	72.3
3K-6K	9.4	81.7
6K-12K	7.1	89.8
12K+	10.1	99.9

Table showing occurrence of various file sizes.

Inodes are stored in the inode table, which is at a location pointed to by the group descriptor within each group. The location and size of the inode table is set at format time, and cannot be changed without reformatting. This means that the maximum number of files in the file system is also fixed at format time. However, each time you format the file system you can set the maximum number of inodes with the -i option to mke2fs.

Directories
No one would like a file system where files were accessed by inode number. Instead, people want to give textual names to files. Directories associate these textual names with the inode numbers used internally by the file system. Most people don't realize that directories are just files where the data is in a special directory format. In fact, on some older UNIXs you could run editors on the directories, just to see what they looked like internally (imagine running vi /tmp).

Each directory is a list of directory entries. Each directory entry associates one file name with one inode number, and consists of the inode number, the length of the file name, and the actual text of the file name.

The root directory is always stored in inode number two, so that the file system code can find it at mount time. Subdirectories are implemented by storing the name of the subdirectory in the name field, and the inode number of the subdirectory in the inode field. Hard links are implemented by storing the same inode number with more than one file name. Accessing the file by either name results in the same inode number, and therefore the same data.

The special directories "." and ".." are implemented by storing the names "." and ".." in the directory, and the inode number of the current and parent directories in the inode field. The only special treatment these two entries receive is that they are automatically created when any new directory is made, and they cannot be deleted.

The File System in Action
The easiest way to understand the EXT2 file system is to watch it in action.

Accessing a file
To explain the EXT2 file system in action, we will need two things: a variable that holds directories named DIR, and a path name to look up. Some path names have many components (e.g. /usr/X11/bin/Xrefresh) and others do not (e.g. /vmlinuz).

Assume that some process wants to

open a file. Each process will have associated with it a current working directory. All file names that do not start with "/" are resolved relative to this current working directory and DIR starts at the current working directory. File names that start with "/" are resolved relative to the root directory (see chroot for the one exception), and DIR starts at the root directory.

Each directory name in the path to be resolved is looked up in DIR as its turn comes. This lookup yields the inode number of the subdirectory we're interested in.

Next the inode of the subdirectory is accessed . The permissions are checked, and if you have access permissions, then this new directory becomes DIR. Each subdirectory in the path is treated the same way, until only the last component of the path remains.

When the last component of the pathname is reached, the variable DIR contains the directory that actually holds the file name we've been looking for. Looking in DIR tells us the inode number of the file. Accessing this final inode tells where the data for the file is stored. After checking permissions, you can access the data.

How many disk accesses were needed to access the data you wanted? A reasonable maximum is two per subdirectory (one to look up the name, the other to find the inode) and then two more for the actual file name itself. This effort is only done at file open time. After a file has been opened, subsequent accesses can use the inode's data without looking it up again. Further, caching eliminates many of the accesses needed to look up a file (more later).

```
Put the starting directory in DIR.
Put the pathname in PATH.
While (PATH has one than one
component)
        Take one component off PATH.
        Find that component in DIR
        yielding the INODE.
        If (permissions on INODE are not
         OK)
                Return ERROR
        Set DIR = INODE
End-While
Take the last component off PATH yielding
FILENAME.
Find FILENAME in DIR yielding INODE.
If (permission on INODE are not OK)
        Return ERROR
Store INODE with the process for quick
later lookup.
Return SUCCESS.
```

Pseudo-code for opening a file.

Allocating New Data

When a new file or directory is created, the EXT2 file system must, decide where to store the data. If the disk is mostly empty, then data can be stored almost anywhere. However, performance is maximized if the data is clustered with other related data to minimize seek times.

The EXT2 file system attempts to allocate each new directory in the group containing its parent directory, on the theory that accesses to parent and children directories are likely to be closely related. The EXT2 file system also attempts to place files in the same group as their directory entries, because directory accesses often lead to file accesses. However, if the group is full, then the new file or new directory is placed in some other non-full group.

The data blocks needed to store directories and files can found by looking in the data allocation bitmap. Any needed space in the inode table can be found by looking in the inode allocation bitmap.

Conclusion

It has been said that one should make things as simple as possible, but no simpler. The EXT2 file system is rather more complex than most people realize, but this complexity results in both the full set of UNIX operations working correctly, and good performance. The code is robust and well tested, and serves the Linux community well. We all owe a debt of thanks to M. Card.

Sources for More Information

The data for the figures in this paper can all be found in my dissertation Improving File System Performance with Predictive Caching. See the URL http://euclid.nmu.edu/~randy .

An excellent paper with more technical detail can be found at http://step.polymtl.ca/~ldd/ext2fs/ext2fs_toc.html .

Some performance data can be found at http://www.silkroad.com/linux-bm.html .

Contact Information

To contact the MUUG board for membership information or anything else, send e-mail to board@muug.mb.ca. We have a Web presence as well, at http://www.muug.mb.ca/, where you can find all kinds of information, including details of upcoming and past meetings and presentations and references related to them. E-mail the editor at editor@muug.mb.ca.

Editor's Notes

The (lengthy) semi-technical article in this issue and much more, including many great tips and other discussions can be found at the Linux Gazette web site at:
http://www.redhat.com/linux-info/lg/

The author of last month's book review of *Linux In A Nutshell* was Kevin McGregor; the byline was inadvertently omitted.