



Figure 1: Matching both forward and backward.

(continued from page 52)

responding smaller chunk in the updated file is enclosed in the larger chunk. One match can enclose another without being expanded. Enclosing matches have unequal distances; expanded matches have equal distances.

The linked list of matches is kept sorted on starting position in the updated file. Before expanding the match, the linked list is checked. If the match is already present in an expanded form, it is not expanded or added; if no expanded form of the match is present, it is expanded and inserted into the linked list. While inserting the new match, all matches enclosed by it are removed from the list. The new match is better because it is bigger. After the match list is built, the complete match list is reviewed. Overlapping matches are cut off, and if a match drops below six bytes, it is removed from the list.

Writing the Diff File

The diff file consists of a header and a sequence of tagged entries. The header contains a signature, file sizes, checksums, and data, specifically for Macintosh files (the type and the creator of the updated file). Some tags are used as headers to data included in the diff file. Other tags encode references to data present in the original file.

Finally, the diff file is written. The tags are four bits in size and are contained in the lower four bits of a byte. The upper four bits, together with zero, one, or two extra bytes, contain a chunk size. There is also a block of bytes for two, three, or four extra bytes to encode a chunk file position within the original file.

A 4-bit field is used to encode sizes 1-16 bytes (size 0 is never used); a 12-bit field for sizes 17-4112 (4096+16); and a 20-bit field for sizes 4113-1,052,688. Larger chunks are encoded by using more than one tag. Depending on the size and location of a chunk, I use either a short or a long code. References to small chunks near the start of the file are encoded in three bytes (4-bit tag and 4-bit size, two bytes for file position). The longest reference to a big chunk near the end of the file can be seven bytes (4-bit tag, 20-bit size, 4 bytes for file position), but most will not exceed six.

The diff file contains two separate, se-

quential diff files. On DOS, UNIX, and OS/2, the second diff file is always empty. On the Macintosh, the file contains the diff information for the resource forks (if present).

Platform Specifics

Macintosh files differ in that they are composed of the data fork and the resource fork. The data fork corresponds to a file on the other platforms. On a low level, the resource fork can be seen as just another file. On a higher level, it is used by the Macintosh OS for maintaining a database-like structure of resources. In BinDiff, the resource fork is viewed as a second file. To keep the code as portable

as possible, I created new versions of standard file I/O functions like *fopen*, *fseek*, *fread*, and *fwrite*. My version of *fopen* has an extra option that lets you specify which fork to open. Macintosh diff files from the data fork are usable on other platforms; those from the resource fork are not.

On many UNIX platforms, a K&R C compiler is available. BinDiff uses double headers: Each function has an ANSI header and a K&R header. If an ANSI compiler is available on a particular UNIX platform, it can be activated with one of the conditional compilation switches. Note that the UNIX version has no progress bar—BinDiff simply displays a message after startup.

Open the door to cross-platform application development with C++/Views.

Liant Software Corporation opens the door to the future with C++/Views™, an object-oriented application framework that gives you full portability across all industry-standard GUI environments.

C++/Views will carry you across the threshold to multiplatform application development, thanks to powerful features such as geometry management, portable resources and a visual development environment, C++/Views Constructor™, for all supported platforms. Our portable class library gives you all the functionality you need to build each part of your native GUI application—from the bottom up. MDI, table, toolbar and bitmap classes are available for all platforms, even OS2/Motif!

- Supports Windows, OS/2, Mac and UNIX environments
- Visual interface builder and class browsing tool
- Class library with over 100 C++ classes
- No royalties or run-time fees
- Source code at no additional cost
- World-class technical support

Call today to find out about our Test Flight evaluation program or to request an information packet and white paper.

LIANT

Phone 1-800-237-1873 or (503) 873-5700/Fax (503) 831-0155
In Europe, phone +44(0) 71-799-2434/Fax +44(0) 71-799-2552

© 1994 Liant Software Corporation. Liant, Liant logo, C++/Views and C++/Views Constructor are trademarks of Liant Software Corporation. All other trademarks are the property of their respective owners.

Putting it Together

BinDiff's complete C source code and project files are available electronically; see "Availability," page 3. I've tested BinDiff with Symantec C++ 7.0 and MetroWerks C++ 1.0 on the Macintosh, Borland C++ 3.1 under DOS, Borland C++ 1.0 on OS/2, and a K&R C compiler on A/UX. For the Macintosh version, you'll also need the file `BinDiff.rsrc`, which contains the resources for a dialog-window layout. The Symantec project file should contain: `BinDiff.c`, `ANSI++.C`, `CPlusLib`, and `MacTraps`. I put the library files in a separate segment. I use `BIDF` as creator and `.DIF` as file type. Because the standard file functions such as `fopen` and `fseek` are already present in `ANSI++`, I define the corresponding functions `FOPEN`, `FSEEK`, and so on, and use macros to convert lowercase functions to their uppercase variants (for example, `#define fclose(x) FCLOSE(x)`). This prevents linking problems.

Using Borland C++ under DOS and OS/2, you must create a project that includes `BinDiff.c`, and change the settings so that `BinDiff.c` is compiled in C++ mode. On most UNIX systems, you can compile with the following command line: `cc bin-diff.c -o bin-diff -lm`.

The source file contains multiple versions of the program; you can change compilation variables according to the version being compiled. By setting `BDEXTR`

to -1 instead of 0, you compile a reduced version of BinDiff, called "BdExtr," that contains only the code for applying a diff file to an original file. Setting one of the values `BorlandC`, `MacC`, or `StdLinuxC` to "1" identifies the compilation platform. The routine `ScanFile` scans through a file and calculates the mean block size and standard deviation for each byte value 0-255 if this byte were to be used as a delimiter.

or twice (for both forks of a Macintosh file). `SubtractFiles` is the highest-level routine to create a diff from an original and an updated file. `AddFiles` is the highest-level routine to apply a diff to an original file in order to create an updated file.

Conclusion

You can optimize BinDiff in several ways. For instance, the unbalanced tree can become a balanced tree. This yields better performance with already-sorted files (such as sorted text files). Next, consider the possibility of using the zero delimiter when no good delimiter is found. The zero default is probably one of the worst choices, but it is very rarely used, normally occurring only on very small files.

Also, more data could be read into memory. Currently, only a very small part of the file is read into the nodes of the tree, making the algorithm rather dependent on disk I/O performance. When a lot of memory is available, more of the file should be read into the tree. Another optimization is to use CRC instead of the simple checksums used for checking the files. CRCs give more security against using a diff on a nonmatching original file, and against diff file corruption. Finally, the diff file could be compressed.

In BinDiff, the resource fork is viewed as a second file

FindDelimiter checks the tables built by *ScanFile* and chooses a suitable delimiter.

BuildTree scans a file and builds the chunk index tree. *ExtendMatch* extends a match forward and backward, until the first nonmatching bytes or the file limits. *MatchFiles* matches the second file to the first file's index tree. *DumpDiff* writes the tagged diff file. Depending on the platform, part of the routine is executed once (DOS/OS/2/LINUX)

DDJ

Selling Software without License Management is like Running a Business without Accounting.



Software Publishers who use FLEXlm in their products provide their customers tools to meter software and ensure compliance with licensing terms. When non-compliant use is detected, developers using FLEXlm can choose to deny service, report use to a log, or simply warn the end-user. FLEXlm offers you a complete range of field proven software or optional hardware-based licensing solutions.

FLEXlm for Developers

- Ensure compliance with your licensing terms.
- Provides network based licensing.
- Implement more effective pricing policies.
- Put protected software on CDs or the internet.
- Activate software by phone, fax or email.
- Make sales and marketing more efficient.

Benefits for Your Customers

- Ensure compliance with publisher's licensing terms.
- Control access to software.
- Capture usage data for internal billbacks.
- Give customers time limited product trials to let them "try before they buy."
- Provides usage data to justify more purchases.
- Easy to use.

Available on Windows, NT, NetWare and UNIX with support for TCP/IP, UDP, DECnet, IPX and "serverless."

Globetrotter Software, Cupertino, California.
Tel: 408-255-5616 Fax: 408-255-6362 email: info@globes.com

De facto Standard License Management for Developers and Systems Administrators



GLOBEtrouter